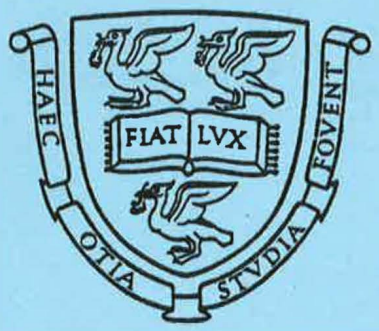
DEPARTMENT OF COMPUTER SCIENCE

ATARI INTERACTIVE
DEBUGGER
by T.P.LOVE
(M.Sc. Dissertation)



The University of Liverpool, Chadwick Building, P.O. Box 147, Liverpool L69 3BX

Contents:

CHAPTER 1

AIMS, REQUIREMENTS AND DESIGN

The basic requirement was clear enough; a debugger for 68k code on the Atari ST to be used by future undergraduates within the department, but it was felt that as much should be done to explore the potential of the WIMP enviroment as possible. Initially I thought that debugging, where one requires fast response to simple requests, and the users are likely to be profficient keyboard users, wouldn't lend itself naturally to such a treatment and I wanted to retain a command line option but as the project rolled on, I thought that I should move with the times and accept some gratuitous wimping for the benefit of those of the intended users who are likely to remain inexperienced 68k programmers.

My part in the production of STID (STID : ST interactive debugger) was to Design the program and do as much programming as I could.

Many of the facilities the program would be expected to provide were standard. I took as a basic model the *ID chip for the 6502 (as presently used in the university), adding facilities that I had found useful while using the MAXAM

monitor/assembler for the Amstrad micro from Arnor and a monitor/ assembler called GEM (no relation) for the Commodore 64. Perhaps the majority of lowlevel programmers nowadays work on home micros and the fruit of their experience shouldn't be ignored. Also there seems to be little difference between 6502 and 68k debugging methodologies. One error that may crop up in 68k but not 6502 code is that of not lining up commands on even addresses. I have tried to ensure that the user is made aware of this risk without imposing inflexibility. For instance, when displaying memory in words they must be displayed from an even address but byte display can begin at an odd one.

I also tried to find out as much as possible about monitors for 68k machines. I found magazine reviews of programs useful but I also looked at some programs that have been used on the Atari, Mac etc. The salient features of these are summarised later.

I consulted articles and books on debugging. The little of relevance that I found is summarised later.

User-friendliness and the exploitation of windows was exellently covered in an article on the Dbx-Tool for the Sun Workstation(1) (They had two engineers working on this problem for months) and in an article on the design of the Star user interface(2). These, augmented by the experience gained by my limited use of MAC and ST programs provided the bulk of my conception of the final product.

C was always going to be the language used. GEM more or less decided that. It took longer than expected for me to come to terms with C and the ST. The books I consulted are listed later. Before I felt I knew what GEM could do, and how fast it could do it, I found it impossible to decide on details of the implementation, so the design and the coding continued in tandem.

Throughout, I was aware that this program would be taken over by others with different ideas to mine so I didn't feel oblidged to specify the operations down to the smallest detail.

First, some general design points.

I decided early on that I wanted an experienced user to be able to use the minimum number of "keypresses" without losing flexibility or reducing the display of options that a new user would need. Using dialog boxes and as many defaults as possible, such an interface is possible. Nearly all facilities provide default values. In deciding what the default value should be, I plumped for uniformity rather than the ideal; it might have been tempting to have the value used in the last call of a function being the default for one function and perhaps 'start of block' being the default for another, but the small theoretical gain in keypresses is easily outweighed by the greater user- friendliness of a consistent default setting.

The debugger should cater for personal programming style, retaining as much flexibility as is tenable. For instance, I wanted the user to be able to keep as many windows open as possible and be able to perform any reasonable inter-window operation. However, I decided not to allow dragging of screen blocks of code/memory (as MacWrite). The risk of user imprecision causing irreversable damage would be too great.

I initially intended using the viewpoint philosophy (i.e. when "looking" at memory you could view it as hex values, 68k code or a dynamic entity whose actions only are visible) but this high-level interpretation of the program makes it less easy to use.

I have neglected color and sound. I'm told that there are spreadsheets that use color and I can see where color coding could be used in a debugger (e.g. when showing memory, ROM values could be printed in black to show that they are "dead", fixed) but the concommitant reduction in effective screen size would be crippling.

Soon there will be 2M and 4M versions of the ST. This affects default values of the top of memory. If chips other than the 68000 are used then the register display will have to be augmented (for details see later).

In line with the prevailing GEM applications philosophy I have not updated every window each time the contents of the active window is changed. By doing this I have allowed contradictory data to appear simultaneously on screen which

shouldn't confuse the seasoned user but may puzzle others.

I have taken the liberty of allowing not more than one label display window and not more than one break point display window to be on screen at a time. My justification of this reduction in the user's freedom is that

i) I can think of no reason why 2 windows of either of these types should be needed.

ii) 'Virtual' inconsistancies are reduced.

Some programs have a notepad option that allows the user to type in passing thoughts. I think that the typical assembler writer works with a listing and various scraps of paper around the keyboard, ticking off corrections as they are typed in etc, and a computer notepad would be superfluous.

Originally, activation of the HELP option was to have caused an explanation of the usage of a facility to be displayed prior to the facility's activation. I was advised that the trend is towards a separate HELP document being on the same disc as the program which can be printed out and refered to when necessary.

I wanted labels to be used whenever possible. Mindful of the fact that debugging a program usually takes more than a session, I thought it would be useful to allow the user to save and load labels. The labels are kept in numerically ascending order. At present there is no sorting procedure; the labels are added and removed one at a time except when loading labels, and

loaded labels are presumed to be in the correct order.

One general feature worth mentioning is that of Highlighting. Whenever sensible, the user is allowed to highlight an address (or a label) and this address would be the default for subsequent commands. In view of the above paragraph it should be noted that at times the user may not know which of the visible highlighted lines is the current one. This may confuse the first time user but I feel the advantages outweigh the disadvantages since it reduces the need to type numbers.

I had considered changing the form of the pointer to some effect. For instance, in a memory display window the pointer could change according to whether the character beneath the pointer was editable or not. I have not implemented this but it could be useful.

Error detecting and reporting must be comprehensive in a program like this, to save irritation on the part of the user (and the harrassed tutor).

Error prevention is included at the lowest possible level (e.g. during the changing of memory only hex characters are accepted. Within dialog boxes such restrictions are impossible to implement so the job is done by glong). I have included many error traps but have been rather remiss in providing informative error messages. SID tries to read from every memory location that it changes and claims to give an error message if the expected value is not read back. In fact it crashes.

The loading and saving facilities will need many error messages.

Now some more specific design details.

There are 2 global variables, 'lblock' and 'hblock' which are initialised to 0x000000 and 0xFFFFFF respectively and take the start and end addresses of any hex file which is loaded in. Having such a default block is useful but if the user wants to look outside of this block then care must be taken by the programmer to avoid silly default values being offered.

Backtracing is a possible addition but is beyond the scope of this project.

It would be useful to be able to put background processes on the interrupt.For example, one could have an updated window on memory showing while a program is being run. This doesn't sound difficult to implement; I just haven't got round to thinking about it.

I have implemented a facility I called 'tagging' whereby suitable windows can be linked so that they scroll together.

A few of the function keys have fixed uses, but most can be redefined so that up to 5 of the menu calls can be held under them. It would not be difficult to save and load the function key redefinitions but it's hardly worth it. The intention was to show the keys on screen (perhaps not permanently) so they could be clicked on.

I have tried to take advantage of what visible structure 68k code has by having an option during disassembly that highlights commands that alter the SP.

At least one 68k monitor allows searching for assembly code instructions with wild fields. I haven't included this facility because I know no one with sufficient knowledge of 68k assembling to tell me whether this facility merits the work involved. No one in the department seems to have done any 68k programming to speak of, which has made me less assured about some of my design decisions.

I have suggested a facility called 'check corruption'. Given 2 addresses, 'start' and 'end' say, this facility starts the program from 'start' and makes a note of which registers have been changed as it passes to 'end' whereupon it reports its findings. It doesn't check whether registers have been saved on the stack.

Monitors on micros often have a code relocation facility. 68k code can easily be written to be relocatable so I haven't included a provision.

Monitors for micros are often relocatable (or at least have 'top of memory' and 'bottom of memory' versions). I don't know how this useful facility might be included in this program.

Some of the TOS calls return information that might be useful to a debugger (e.g. where the top of memory is). I haven't included a provision to call these from within the

debugger. Such a provision would sometimes be useful but to be fully implemented would require a 'command line' style of entry with TOS documentation near at hand.

Screen and Menus: The menu bar is DESK, GO, LOOK, ADD, BLOCK, SET, TRANSPUT

DESK:(contains desk accessories and Exit)

GO: (Dynamic Features)

- Go
- Trace
- Set Breaks
- Show Breaks
- Check Corruption

LOOK:(Look and Change)

- Same?
- Disassemble
- Find
- Replace
- Inspect
- Registers
- Graphics

ADD: (Extra Code)

- Assemble
- Patch Insert
- Preset Patch Area

BLOCK:(Large scale changes)

Fill

Copy

SET:(Setting up formats etc)

Help

Base

Define label

List labels

Trace Format

Block Set

TRANSPUT:

Redirect

Save

Load

Delete

Save Labels

Load labels

Function Keys:

Redefinable (Soft)

f2 Set Break

f3

f4

f5

Fixed

f1 Start redefine

f6 Cont (After a break,continue)

f7 Repeat (Repeat the previous menu call)

f8 Tag

f9 Untag

f10 End redefine

****

CHAPTER 2

SPECIFICATION

Specification of the user-visible menu functions and some lower level routines to simplify the descriptions.

0. DATA STRUCTURES and main variables etc

```
WI_KIND1
= (SIZER'MOVER'FULLER'CLOSER'NAME)
WI_KIND2
= (SIZER'MOVER'FULLER'CLOSER'NAME'UPARROW'DNARROW'VSLIDE)


LAB_LENGTH =8

onoff[0]="off";onoff[1]="on"
int phys_handle
int handle
int xdesk,ydesk,wdeck,hdesk
int msgbuff[8]
char string[80]
char target[80]
```

```
char wild[80];
char chbuff[80];
char numbuff[80];

int no_of_brkpts
int first_brkpt

int no_of labels
int first_label

int no_of_windows
int max_no_windows
int stack[3];

long lblock, hblock, lblock1, hblock1, lblock2, hblock2
long bl_size

long ltemp
long total

char * addr,addr1


int top_handle
int topwnum

long hilite_addr
int hilite_on

int tag
int button
```

```
int redef
int newkey

int no_of_formats=1
int pres_format=0
int base
int bytes_per_line
int bwl (byte,word or long)
int digits

struct labelitem .char name[LAB_LENGTH]; long num;
struct labelitem labels[50];  Kept in inc. numerical order

struct brkitem
 .char name[LAB_LENGTH]; long num;int count,on,saved_word;
(note that the labelitem structure could be used for the
  first two fields. )
struct brkitem brkpts[50];   Kept in inc. numerical order

struct formatitem .char name[LAB_LENGTH];long flags;
struct formatitem formats[10]
       format[0]=( "Registers",ffffffff)
(I proposed to used bit flags to show which of the 16
 working registers and other 68k registers were selected
 for that format. This may be unnecessarily niggardly on
 memory. I don't know how easy bit manipulation is in C.
 Note: The 68000 has 2 A7 registers.
       The 68010 has a vector base register and
           2 alternate function code registers.
```

The 68020 has a cache control register and
a cache address register.)

```
struct windowitem
 .int xwork, ywork, hwork, wwork, xold, yold, hold, wold,
       on, type, han, fulled, bwl, tag;
    long loc;
```

('han' is short for handle. Note that not all the fields are used for all windows. E.g. 'bwl' shows whether a memory window is displaying byte, word or long values.)

```
struct w[3]

struct fn_key .int on,num_fns, (*fns)()[5];
struct fn_keys[10]
```

1. MAIN SUBROUTINES

g_long_(string)

Octal,decimal and hex numbers are accepted.

Label inputs are accepted.

A long is returned into 'total' and TRUE returned,
else FALSE.

****

int find(target,wildmask,from,to,&addr) looks for the target array of values in the block from-to.

If nothing is found it returns 0 else it returns 1 and the address.

```
*****************************************************

add_label (&temp_lab_item)
    if the labelname exists, delete old item, no_of_labels--
    check if already max no_of_labels.
            If so, print warning and return(0)
    Find correct insertion point for new label;
            make space for if;
            no_of_labels++;return(1)

*****************************************************
print_addr(x,y,addr)
         This prints the address, or the first label
         in the label list with this value, at
         position x,y on the screen.
*****************************************************
```

2. FUNCTIONS

****EXIT****

0. (possibly) If no code or no labels have been saved then warn.
1. Close all windows
2. Blank out the menubar
3. Close workstation etc.

*****GO*****

0. Open dialogue box
1. last_exe_addr = g_long
2. Call a 68k routine
    . Save regs
    . Set regs up as they were last seen/set by user

. Save the interrupt mask

. Reset the USP ?

. Set up brkpts

. Put PC onto stack

. put last_exe_addr into PC

. Restore PC

. Restore brkpts

. Restore mask

. Restore regs

3. End

SHOW BREAKS

0. Open a window type WI_KIND2

1. Using sprintf("%8s %1 %d %s",

brkpts.name[i],brkpts.num[i],

brkpts.count[i],onoff[brkpts.onoff])

print the breakpoint information,

starting with first_brkpt

2. Set break_flag.

3. Await window/menu event,'return' or mouse button.

If mouse pressed on name,value or count

then allow change. Beware of name duplication.

If mouse is double clicked on a line

then remove the brkpt from the table,

decrement no_of_brks and refresh window.

Beware of removal of the only brkpt.

SET BREAK

0. Open dialog box

1. Await inputs

temp = g_long;

default count=1

2. If temp is odd,

give an alert and return to stage 1

else last_brk=temp

3. Add new brkpt to the table and reorder.

Increment no_of_brks

4. End

****TRACE****

0. Open dialog box

1. Await inputs

trace_from=g_long (default trace_from)

trace_type= 1(STEP) or 2(FAST) (Radio buttons)

trace_depth=0(SHALLOW) or 1(DEEP)

2. Open an output window

3 ???

*** CHECK CORRUPTION /WATERGATE ***

0. Open a dialog box.

1. Await inputs

start=g_long

finish=g_long

2.???

****FIND****

0. Open dialog box

1. Await inputs

start_block =g_long with default BLOCK;

end_block =g_long with default BLOCK;

target is an array of values

This can be a string, introduced by a quotation mark or a set of hex bytes.

When the 2nd digit of a 2 digit hex value is typed in, the cursor skips a space.

Wild cards are allowed.

Within a string, '?' represents an unspecified character.

When a '?' is typed in a hex input the 2-byte cell is taken to represent an unspecified value.

2. If output is to screen, open a full screen window.

3. Using the format of 'inspect memory'
print the target bytes as they are
found, together with some neighbouring bytes.

/* NOTE: INCLUDE ABORT PROVISION */

REPLACE

0. Open dialog box

1. Await variables

(as in FIND, except for 'new' and 'confirm y/n')

2. Check that len(target)==len(new).

If not, then return to step 1

3. temp=start_block

while(find(target,wildmask,temp,end_block,&temp))

do . if(confirm) THEN REQUEST CONFIRMATION

If all's ok,

replace the instance of 'target' by 'new'

CHECK SAME

0. Open dialog box
1. Await variables

(default source is BLOCK,

default dest_start is the

current highlighted line, if any)

2. Open a window WI_KIND1 for output
3. Check same. Print differences.
4.End

/* INCLUDE ABORT PROVISION */

**** INSPECT ****

0. Open a dialog box
1. Await radiobutton ByteWordLong

and from= g_long selection

2. If Word or Long and odd addr

go back to stage 1 with a warning

3. Close window.
4. Open a WI_KIND2 window
5. Display addr,bytes/words/longs,chars
6. Await window/menu and button events

byte values and chars can be changed.

DISASSEMBLE

0. Open a dialog box.

1. Await variables:

from=g_long default is highlighted line

else last_dis_addr

line numbers(y/n) default NO

highlight option(brkpts,stack ops,none)

default is last choice or NONE

2. If from is odd, return to stage 1 after a warning.

3. Close box

4. Open a WI_KIND2 window with info line containing headings.

5. Print

¼ line no.½ ¼ label½ hex addr source code hex

dec 4 digits LAB_LENGTH 6 digits 20 chars 14 digits

if hex addr = a brkpt value and option=brkpts or

if command is RTS RTI JSR,etc and option= stack ops

then Highlite the output line.

The addresses to which jumps and branches

go can be printed using

'print_addr' in the source code field.

*** Remember to decode the Axxx commands ***

6. Await menu/window and mouse button events.

Clicking on a hex byte will allow it to be edited

resulting in a window refresh.

Clicking on the addr will make it

the current value of 'hilite'.

Note that movement of the scroll bar

should never allow display starting

from an odd addr.

REGISTERS

0. Open window WI_KIND1
1. Display 17 regs, flag fields and a display box with a default register value in current base, all other register values in hex.
2. Clicking on flags, base, will cycle thru possible values.
3. Clicking on a reg value puts the contents into the display box and allows editing of the number in the display box or in register

Register Display

Registers

D0	01 0a ff 57	A0	01 33 e5 ff
D1	01 0a ff 57	A1	01 45 e5 ff
D2	01 0a ff 57	A2	01 65 f4 ff
D3	01 0a ff 57	A3	09 77 b6 ff
D4	01 3d ff 57	A4	08 66 b4 ff
D5	01 6d ff 57	A5	88 77 c6 ff
D6	01 7a ff 58	A6	99 57 f1 ff
D7	01 7d ff 59	sA7	
		uA7	88 33 d9 ff

Flags

1	Trace
0	Supervisor
5	Interrupt.
1	X
0	N
1	Z
1	V
0	C

N

Mode A6

bin	1001 1001 0101 0111 1111 0001 1111 1111

Preset Patch Area

0. Open dialog box.

1. Await variables

patch_start = g_long()

patch_end = g_long()

2. If the defined block overlaps ROM, STID or Block
then issue warning.

3. End

ADD

(This facility allows the writing of simple 68k code)
(which will go into the next piece of free space in)
(the patch area. I'm leaving this for those who)
(programmed similar things before)

PATCH AT

(This call is used to show where the patch is to be)
(called from the main sequence.)

****FILL****

0. Open dialog box

1. Await variables

(as in 'find' except no wilds in the input string)

2. Fill

3. End

COPY

0. Open dialog box

1. Await variables

   (default block to be copied is BLOCK

   default dest_start is highlighted line, if any)

2. Do intelligent copy

3. End

BLOCK SET

0. Open dialog box

1. Await variables

2. Check that start_block ¼ end_block. If not, go to stage 1.

3. Reassign start_block, end_block

4. End

*****BASE*****

0. Open dialog box with radio buttons.

1. Await 'ok'

2. Reassign 'base' to 2,8,10 or 16

3. End

DEFINE LABEL

0. Open dialog box.

1. Await a string 'labelname' and a value

   (which could, I suppose, be a label)

2. Call add label.

3. Close dialog box

*****LIST LABELS*****

0. Open a window WI_KIND2

1. Print the labels in numerical order using

sprintf("%8s % .default base: ",
labels.name[i],labels.num[i])
starting from first_label

2. Set lab_flag.

3. Await window/menu operations and mouse button.
If mouse pressed on name or value
then allow change. Call add_label.
If mouse double clicked on a line,
remove label from table,
reorder,
no_of_labels--
Beware if the only label is removed

*** So how can a line be hilited? ***

*** COMMAND LINE ***

???Maybe this is unnecessary???

*** TRACE FORMAT ***

0. Display the names of trace formats .
Highlight the present one.

1. Await selection.

2. If a new name, check to see that max not exceeded;
add name to formats.names with flags set to ffffffff
format++;pres_format=selection

3. Close display

4. Display, fullscreen, the Registers of selection with
non-selected items blocked out. Print the format_name.

5. Allow clicking on Regs to de/select them.

*** TABULATE ***

0. Open Dialog box
1. Await variables

   Yes/No

2. If Yes then all subsequent inspect and disassemble output will be arranged so that no information is lost off the edge. In inspect() this is acheived by adjusting bytes_per_line.

LOAD LABELS

0. Open a window showing available .lab files using fsel_input
1. Await filename selection.
2. Load the file into memory represented by 'labels'.
3. Reset first_label to zero.
4. Count thru the labels until
   there is a label field full of 0xFF bytes
   (denoting end of labels) and reset no_of_labs.
5. If there's an open label window
   then activate it and 're_draw' it.
5. Close window.

**** SAVE LABELS ****

0. Open a window showing available .lab files.
1. Await filename selection.
2. If chosen filename is already being used,
   then erase the file.
3. Save the file from memory represented by 'labels'.

4. Close window.

****LOAD****

NOTE: If you have a problem with the restoration of the screen after an fsel() call then consult AUG '86 BYTE, Atari notes.

0. Open a WI_KIND2 window to show all relevant filenames using fsel_input.
1. Await selection of a file and load_addr= g_long
2. Check load_addr (and length of file ?!) to ensure that sensitive memory (eg stid space) is not overwritten.
3. Load.
4. Update block_start,block_end
5. Close window.

*** DELETE ***

0. Open a window using fsel_input to show all files on the default disc.
1. Await file selection.
2. Ask for confirmation (The delete,load and save screens are visually similar)
3. Delete the chosen file.
4. Close window.

**** SAVE ****

0. Open a WI_KIND2 window to show .hex files
1. Await selection, start= g_long (default block_start)

end = g_long (default block_end)

?! exe = g_long

2. If not a new filename,ask confirmation,then erase old program.
3. Save new program.
4. Close window.

**** REDIRECT ****

0. Open a dialog box.
1. Await 'printer' or 'screen' radio button selection.
2. Exit.

**** GRAPHICS ****

0. Open a dialog box.
1. Await variables.

   Start Address

   Width (pixels)

   Height (pixels)

2. Open a WI_KIND2 window to show how the block of memory looks in high resolution graphics.

CHAPTER 3

WINDOWS

WINDOW MANAGEMENT

The info about the windows is kept in a struct called w.

There is a (sort of)stack containing the struct indices of the windows, ordered by depth, so stack[0] contains the struct index of the top window.

WINDOW CREATING

Each option requiring a window needs its own routine to create a window, entitle it etc.

WINDOW OPENING

One routine can open all windows as long as an adequate window positioning stategy can be included in it. Minimal overlap (also known as tiling) has the advantage from no information or corners are eclipsed, but overlapping is often more convenient for the user. The present 'strategy' is that all windows are half screen size, except for label windows which are set to minimum size ,and they appear on alternate sides of the screen. The stack routine 'newwind' should be called.

CLEAR WINDOW

Clears the window to white.

MOVE/CHANGE SIZE OF WINDOW

Standard. See any GEM book

TOPPED/NEWTOP

Standard. The stack routine 'topper' should be called.

FULLED

Standard.

REDRAW AN AREA

See program.

CLOSE TOP WINDOW

Standard. The stack routine 'closetopwindow' should be called.

THE STACK

Stack management is performed by 3 routines:-

newwind:- Puts the struct index of a newly opened window onto the top of the stack.

topper:- Puts an existing window's struct index on top of the stack.

closetopwindow:- Removes the top of the stack.

CHAPTER 4

CODING AND TESTING

I inherited from the program development package a program which included routines for moving and changing the size of a single window.

First I added a menu and extended the program to handle 4 windows.

Then, being more aware of GEM capabilities and having decided what the user should expect from the system I wrote a specifications document describing the user-visible routines.

As I programmed using GEM, the specification changed. The structure only 'emerged' as the program grew.

I began designing the program from the top and coding from the bottom, trying to keep the scope of variables as restricted as possible and trying to emphasise modularity though it must be remembered that adding a new facility requires additions to be made at various places in the program. The necessary bottom level functions emerged during the design phase and were coded first as stubs then as restricted versions, adequate to test the higher level functions.

The decision as to which functions to code was made according to these criteria:

1. The need to check speed/feasibility of functions.

2. The desire to use a wide spread of C facilities.

3. The desirability of producing an integrated suite of functions.

Some functions are completely coded, others are unstarted. To cater for the different sort of documentation that this range of completion requires I have chosen a pseudocode representation, allowing flexibility of depth of detail.

As an example of my testing style let's suppose I discover during a run that the 'find' facility doesn't work. Now there is one level above (the command,menu, function key handling) and 2 levels below 'find' ('findit(...)' and 'glong()' are just below and 'glong()' calls on 'val()).

'val()' is simple and short, 'glong()' is a simplified and up to now unfailing version of the final routine, it only being able to handle labels and hex numbers and the top level is unlikely to be the culprit so the error is likely to be in the 'find' routine itself, where most of the code is. Having localised the source of the error as far as possible I would add some print statements to help confirm or deny a hypothesis. The output of these can often mess up the GEM display so they should

be used sparingly.

If all this fails I would put the offending code into a little program of its own and work on it further.

CHAPTER 5

PROGRAM NOTES

PROGRAM NOTES

Here I have tried to give a general overview of the program then some helpful, informal notes to those who will, I trust, continue the project. This part of the write-up, even moreso than the other parts, is written for my successors to the project. It is to be read beside an Atari loaded with STID and a list- ing of the program. It should help you find your way about the program. At the end is a list of bugs and places where work is needed.

Most of the work is done in multi() which looks for events using a switch. The consequences of these events depend on the type of contents of the active window, necessitating other switches, so the final shape of multi() is almost grid-like. Adding a new facility often means adding to many parts of the program.

Declarations:-

Objects:-

Dialog boxes: find,replace,inspect,fill,copy,same

lab

General, low-level graphic routines:-

hide_mouse

show_mouse

open_vwork

set_clip

do_redraw

open_owindow

open_ewindow

open_mwindow

open_lwindow

openw

close_wi

cls

dialbox

main

multi:- (See later)

Highlevel routines:-

draw_ellipse

showmem

showlabs

find

replace

inspect

fill

copy

same

def_lab

loadhex

Window Stack commands:-

close_top_window

topper

newwind

General, lowlevel nongraphics routines

inv_color

locator

value

wildstr

wildbyte

findit

g_long

block2

find_vacant

add_label

chop

cursor

count_labs

num_alert

wind_alert

lab_alert

```
        sh_lab_alert
-------------------------------------------------------
multi:-
        mn
        redraw
        arrowed
        vslid
        newtop,topped
        sized,moved
        fulled
        closed

        tag check
        last command check
        click in window check
        check function keys
-------------------------------------------------------
```

Some "informal" program notes, put here because they need only be read once and would clutter up the program.

** Dialog boxes:

Need some tidying up. The input lengths are presently rather arbitrary. Hex inputs for search/replace options are not presently supported.

Not all the suggested defaults for inputs have been implemented.

It's a pity that GEM sets the cursor to the end of the string in an editable field.

All labels need to be distinguished from numbers and string by a preliminary quote mark. A final quote mark is optional.

** doredraw:

Beware. This doesn't work correctly.

** open_owindow
** open_ewindow I decided to keep the window
** open_mwindow parts of the routines separate
** open_lwindow

** openw:

All window opening passes thru here so this is where an algorithm to place the windows to maximum advantage should go. So far only label windows are treated so that they are opened at the minimum size that hides no information.

** dialbox:

** draw_ellipse:

Only kept in for test purposes.

** showlabs:OK

** main:

All system initialisation is left to appl_init, which should close all windows etc.

** find:

This can't handle output that overflows the window. Two things can be done to solve the problem; overwrite old output lines or save the locations of the targets in readiness for scrolling the window. There needs to be an Abort facility rather than just a Continue one after a line of output has been printed.

** replace:

Presently only works for strings.
Perhaps a check should be made to see if
the memory has been successfully updated.

** inspect:OK. See Showmem.

** fill:

Presently only works for strings.
Perhaps a check should be made to see if
the memory has been successfully updated.

** copy:

Intelligent copy.
Perhaps a check should be made to see if
the memory has been successfully updated.

** same:

This can't handle output that overflows the window. Two things can be done; overwrite old output lines or save the locations of the targets in readiness for scrolling the window.

There needs to be an Abort facility rather than just a Continue one after a line of output has been printed.

** def_lab:

Presently allows definitions of strings that begin with a digit.

** load_hex: Only the fsel_input call is here at the moment.

Window Stack commands:-

close_top_window

topper

newwind

General, lowlevel nongraphics routines

inv_color

** locator:

The screen editor for memory dump. The code here could be sharpened up perhaps, but it is the part of the program most nearing completion. Two possible, easy improvements are:

The ability to move the cursor between hex

and ascii

The ability to move the cursor to the rightmost visible hex digit. Presently the program only allows you to move into a block of hex digits if the whole block is visible.

No messages are returned if the user tries to change ROM, an I/O port or non existant memory. Perhaps a check should be made to see if the memory has been successfully updated.

The cursor doesn't loop around at the end of a line. This is easily implementable. The window doesn't scroll up when an attempt is made to move the cursor up from the top line. Similarly for the downward direction.

value

wildstr

wildbyte

findit

** g_long:

This presently only accepts short labels and hex numbers.

block2

find_vacant

add_label:

Not tried out.

chop

cursor

count_labs

num_alert

wind_alert

lab_alert

sh_lab_alert

---

multi:-

** mn:

None of the menu items are disabled during use yet. These are some possible events where this needs to be added to the program.

'Assemble' should be disabled until a patch area has been set.

'Save labels' should be disabled until at least one label exists.

do_redraw

arrowed

vslid

newtop,topped

sized,moved

fulled

closed

** tag check:

The original intention was this:

If F8 was pressed then the active window (if there was one, and if it was of type MEMORY or DISASSEMBLE) would be tagged to the next window of type MEMORY or DISASSEMBLE activated. Henceforth, whenever a tagged window is scrolled by clicking on an arrow, the other tagged windows scroll too. I haven't decided what would be the best thing to do if the scroll bar of a tagged window was operated; it may be best to leave the other tagged windows unaffected.

At present, if the F8 button is pressed then all existant memory windows are tagged together.

last command check

** click in window check:

This has to look out for the various intentions of the user: highlighting, changing memory etc

** check function keys:

There are at present no messages displayed during function key redefinition. The original intention was to do this:

Have the buttons represented on screen with their abbreviated functions.

If F1 were activated, show alert box "please redefine function key 2,3...or 6 or Cancel" and change the button text from "REDEF" to "ENDDEF".

Disable menu commands until another function key is pressed. (not strictly necessary but advisable).

Show the functions presently under the chosen key. Clicking on a function would remove it from the list, clicking on a menu item would add the revelant command to the list.

F1 re-activated would signal the end of key redefinition.

I have only implemented this in skeletal form. To define F2, say, so that pressing it displays the labels, the breakpoints and then the dialog box for inspecting memory, this is what must presently be done.

Press F1

Later, but not necessarily immediately after, press F2.

Click on 'show labels', 'show breakpoints' then 'look'.

Press F10 (note: not F1 because that would have meant checking that the button had been released before being pressed to signify another redefinition. Using F1 is neater from the users point of view.)

BUG LIST

1.Sliders crash. Arrows don't. I haven't detected clicking on the scroll bar that isn't on the slider.

2.The double showing of the contents of the first window remains mysterious.

3.Having hit return after editing memory another cursor appears instead of the expected pointer. This is a problem that my tutor knows the answer to.

CHAPTER 6

CRITICISMS:-

I should have found more 'rival' programs. I could only find brief write-ups of them.

The memory display is too slow. This may be because the number of bytes displayed per line is not fixed, so the format of a line is not fixed, being built up unit by unit within a loop. Scrolling of windows could be accelerated by blitting but for the fact that a tagged window may be partly obscured by another. The top window could always be blit-scrolled though.

The design shouldn't need radical changing though some routines may need an extra parameter added to them to make them more general. The design isn't complete, neither is the program. Of course, the program couldn't be finished in the time but the design could have been. I had to decide what the purpose of the programming aspect of the project was. I could merely have used programs to check the feasibility of design ideas. I decided to use programming as a source of ideas as well as a testbed and a way to broaden my program experience. To have completed the design having done little programming in GEM and having done no 68k assembler programming let alone

debugging would have been at least pretentious.

The suggestion of my putting shells and comments in the program to show how the final program would work did not receive encouragement from my tutor.

The choices that I have made for the defaults may not be optimum. In the program I have never used as a default parameter the parameter used on the previous call of that facility. The best way to make such choices is by experience.

Labels need a quotation mark before them to distinguish them from numbers, but when there is no risk of confusion (e.g. when defining a label) , the quotation mark is not required. This is an inconsistancy.

The window refreshing problem really should have been sorted out earlier. I lived with this problem, being unable to solve it. This illustrates two other points:

1. At times I lacked the imagination to come up with hypotheses to base diagnostic investigations on. If I followed the book and it didn't work I didn't do any explorations. I asked other people and if that got nowhere I just left the problem.

2. I acted as if I were part of a team effort even tho' I knew the rest of the team would only start, if at all, after I had finished. Some of the evident lack of thoroughness results from my suggesting something and not having the self- confidence to be definitive without feedback from

team-members-to-be.

CHAPTER 7

REFERENCES ETC

*** Present 68k Monitors/ Debuggers *** 1. TUTOR (Described in "The Motorola MC68000" by Jean Bacon. It is the monitor that comes free with the 68k development board)

A non-graphic program. The only notable feature is that the Memory Display option asks whether you want a hex or 68k output.

2. SID (Described in the Atari documentation)

Occupies the top 20k of TPA. It allows labels, (preceded by '.') and is non-graphic.

The facilities are a subset of those suggested for STID save for a "load and run" option and 2 facilities whose descriptions I supply here.

" The I command prepares a file control block (FCB) and command tail buffer in the base page of the last file loaded with the E i.e. "load and go" command. The form is as follows:

Icommandtail

where commandtail is the character string which usually contains one or more filenames. The first filename is passed into the default File Control Block at 005Ch. The optional second filename, if specified, is passed into the second default File Command Block beginning at 0038h. The characters in the command tail are also copied to the default command buffer at 0080h. The length of the command tail is stored at 0080h followed by the character string terminated with a binary zero."

"V This causes... the starting address and length of each of the segments contained in the file, the base pointer, and the intial stack pointer to be displayed. "

The documentation also has a list of error messages which may be useful checklist for those writing the file handling part of the program.

---------- References, Atari Technical Data, current debuggers and notes about the utility programs

*** PERIODICAL REFERENCES ***

1."Dbx Tool- A Window based symbolic debugger for the Sun Workstation" in Software Practise and Experience V16(7) p.653 July '86

This is based on UNIX 4.2 BSD dbx and can handle C, Pascal, Fortran and assembler. The article concentrates on the user interface, to which much time was devoted. These desirables emerged from a study:-

NEEDS: Scrolling window for source text.
Mouse driven
Graphical display of breakpoints, execution locus.
Dynamic values of variables

EXTRAS: Should source window be part of debugger?
Should all commands be mouse-constructable?
How should variables be displayed?
How should the command dialog ensue?

They used 5 screen width, variable height windows:
STATUS (giving filename,range of line numbers),
SOURCE
BUTTONS (rather than popdown menus, because a few commands were found to be used most of the time)
COMMAND (button commands echoed)
DISPLAY (vaiable values etc displayed)

The button window was scrollable and extra buttons could be added. The 6 most common functions were found to be:- PRINT, STEP/NEXT, STOP AT(Setting a breakpoint), CONTINUE, STOP IN (Setting a breakpoint at the start of the procedure that the cursor is in), REDO.

The user selects an option by clicking on a button then provides the parameters by pointing to the source code.

The article provides an annotated, illustrated walkthru of a short session.

2 similar programs are:- BugBane in the Cedar Enviroment at Xerox PARC (details available in a research paper). Joff, a debugger for the BLIT terminal at ATT, Bell (details in Tech. Journal 63(8))

---

2. "Designing the Star User Interface" (V7 No.4 BYTE Apr '78)

The mouse is considered to be better than a light pen or tablet because

1. It provides as easy and quick a way of pointing as a finger.

2. It stays where it is if left untouched.

3. It has buttons.

The main principles pursued in designing the interface were:

1. Familiar user's conceptual model.

2. Seeing and Pointing is preferable to Remembering and Typing.

3. WYSIWYG.

4. Universal commands.

5. Consistency.

6. Simplicity.

7. Modeless Interaction.

8. User tailorability.

Few would argue with this list, but the ruthless implememtation of these aims is not easily done. In addition the system should minimize the need to learn or memorize and should give feedback and have a symmetrical command structure.

---

3. "A tour through Cedar" (IEEE software 1 ('84)) by Teitelman

Issues addressed here are:-
Default window placings
i) Tiling the screen allows maximum readability.
ii) Overlapping the windows gives the user flexibility and access to more onscreen information but can lead to "lost" underwindows, wasted screen space or covered window corners.

The importance of command interface uniformity.

---

4. "A fundemental approach to debugging" by Tratner in Software Practise and Experience V9, p.97 '79

During debugging one should:- Reduce uncertainty, Record history, Exert controls, Examine/Modify values.

---

5. "ADDS: A dialog development system for ADA" by A.Burns and J.Robinson. in J. of Man-Machine Studies V24 p.153 '86

The authors suggest that:- Help should be sensitive to context and user-experience.

Backtracking, allowing the reversal of past actions, would be useful.

Interfaces should be adaptable and multi-level. Experts' needs (Minimal information Redundancy, Minimal keypresses) may not be the casual users' needs.

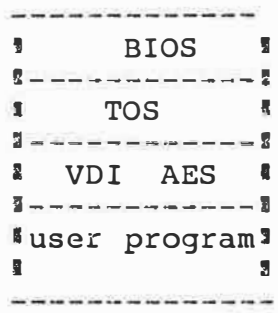
---

6. "Pict: An Interactive Graphical Programming enviroment" by E.P. Glinert and S.L.Tanimoto in IEEE Computer '84

What is needed is a representation that resembles the users' thought processes. The emphasis should be on accuracy, feedback and directness so that the user feels that s/he is "doing" rather than "telling".

2 unobtainable PhD theses are mentioned, both of which describe programs written in Smalltalk and using Icons.

*** NOTES ON THE ATARI ***



The TOS consists of GEMDOS from Digital Research.

XBIOS routines are called by putting the function number and parameters on the stack then tripping a TRAP 14. Some useful calls are:

#2 returns base of physical screen RAM

#20 does a screen dump

GEMDOS routines are initiated by a TRAP 1. Some example call are:

$20 get into supervisor mode

$30 get version number

BIOS routines are initiated by TRAP 13

An assembler Axxx call creates a trap, giving 14 fast,primitive graphics routines.

The ROM lies between FC0000 and FEFFFF.

A few useful locations are:-

00 SP boot up value

04 PC boot up value

$42e phystop (=RAM+1)

$434 memtop (usually (42e)-32k

$44e base of screen memory (usually $f8000 on the 1040ST. Notice that the uppermost 768 bytes (32k-32000) are free to squirrel little 68k routines in.)

To access the system variables you must be in Supervisory mode.

*** BOOKS ***

Programmer's guide to AES and VDI- D.R.

The Anatomy of the Atari ST -
Geritis, Englisch and Bruckman (1st Publishing)

GEM on the Atari -
Szczepanowski and Gunther (1st Publishing)

Tricks and tips of the Atari ST (1st Publishing)

The Atari ST explored -
J.Braga (Kuma Computers)

The C toolbox -
W.J. Hunt (Addison-Welsey)

C:A Reference Manual -
S.Harbison and G.Steele Jr (Prentice-Hall)

The complete guide to writing Software User Manuals -

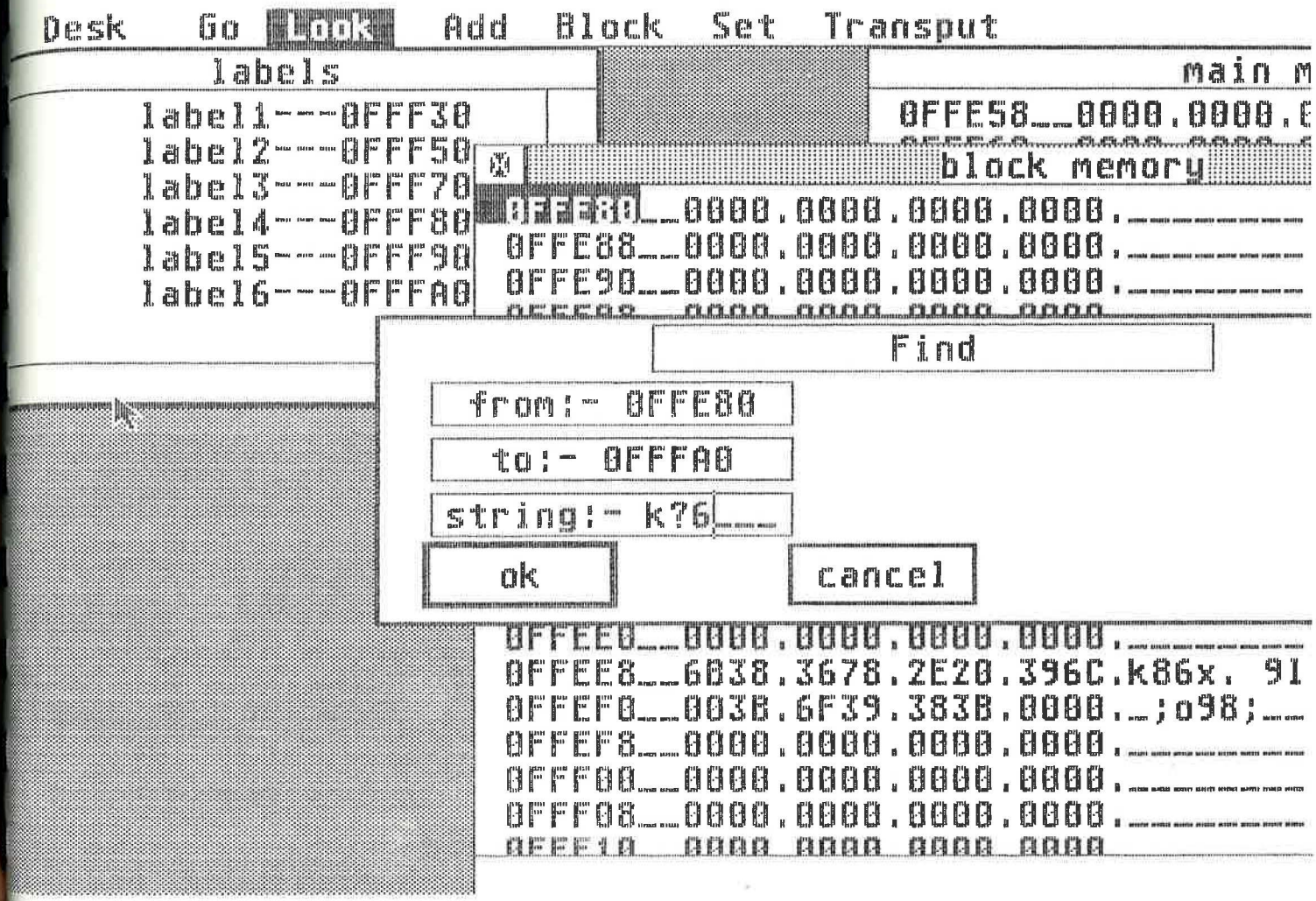
B.M.McGehee.

*** UTILITIES ***

MicroEMACS editor: Non GEM. Has most of the required facilities tho' a TO LINE ? would have helped and some facilities (eg Merge) aren't explicitly mentioned.

Digital Research C compiler: A 3 pass compiler which produces 200k intermediate files from 50k programs. It needs far fewer CAST's than Lattice C for example.

Atari Resource Construction Set:We used version 1 which had no documentation and was crashable. I only used it when I was confident that it wouldn't let me down.


Desk Go Look Add Block Set Transput
labels
main M
label1---0FFF30
label2---0FFF50
label3---0FFF70
label4---0FFF80
label5---0FFF90
label6---0FFFA0
0FFE58__0000.0000.0
block memory
0FFE80__0000.0000.0000.0000.
0FFE88__0000.0000.0000.0000.
0FFE90__0000.0000.0000.0000.
Find
from:- 0FFE80
to:- 0FFFA0
string:- k?6
ok
cancel
0FFEE0__0000.0000.0000.0000.
0FFEE8__6038.3678.2E20.396C.k86x. 9l
0FFEF0__0038.6F39.383B.0000.;o98;
0FFEF8__0000.0000.0000.0000.
0FFF00__0000.0000.0000.0000.
0FFF08__0000.0000.0000.0000.

CHAPTER 8

HANDBOOK

This program is an interactive debugger for 68000 code on the Atari STF1040.

It uses Trace and illegal instruction exceptions, so beware if your program uses them. It saves the state of the interrupt mask prior to running a program and restores it afterwards.

Windows are extensively used. For instructions on their operation see the Atari manual.

"Clicking" unless otherwise stated means pressing the left button of the mouse.

The program is designed so that key/button pressing is minimised. Default values for input parameters are provided wherever possible. Many operations work within a default block of code that can be reset by the 'block set' command.It is initialised to the whole memory size, then set to the size of a loaded hex file.

Wherever reasonable, labels and addresses can be clicked on, which will cause them to change to inverse video. Henceforth this value will be used as the default address for facilities. The line becomes de-highlighted by i) reclicking on it ii) closing the window iii) highlighting another line. If neither of these above possibilities are relevant then the value of the parameter used in the previous call is the default value.

The default base is hex but binary,octal and decimal numbers are supported. E.g. %1101 ,015, ,13, $d all represent the same number.

Labels can substitute for values. They must appear in quotes (the final quotation marks are optional) and can be up to 8 characters long.

The function keys are used, but can also be activated using the mouse. Keys f0-f7 can be redefined to perform one or more of the menu functions as follows:-

1. Hit/click REDEFINE button. The button will change to read ENDDEFINE.
2. Hit/click the button to be redefined. A box showing the curent function of the button appears.
3. Clicking on a menu item will cause this to be added to the functions. Clicking on a function in the box removes it.
4. Repeat step 3 until redefinition is complete then hit/click the ENDDEFINE key.

Wherever it makes sense, windows can be "tagged" so that they scroll together. This is done as follows:-

i) Activate one window.

ii) Hit/click the TAG button. (F8)

iii) Activate another window.

More than 2 windows can be so tagged. Hit/click UNTAG or close a window to cease tagging.

If ever you want to repeat the previous command then press F7.

LOOK

This displays memory in hex and ascii form, and allows modifications to be carried out.

The window can be manipulated as usual. To change the contents of memory, click on the character to be changed. The pointer will dis- appear, to be replaced by a cursor which can be moved using the arrow keys. New values can be typed in. The pointer can be recalled by pressing RETURN.

If the displayed memory is wholly within the block then the vertical slide bar size and position is relative to the block and the window is headed 'block memory' else the scroll bar is shown relative to main memory.

FIND

All occurrences of the target string within the specified block are reported. Wild characters in the target string are represented by a '?'.

REPLACE

All occurrences of the target string within the

specified block are replaced by its substitute. Wild characters in the target string are represented by a '?'.

FILL

The specified block of memory is filled with as many consecutive copies of the string as possible, the remainder being filled by a part of the string.

COPY

One block of code is copied onto another. Only 3 of the 4 block enders need to be legal; the other is deduced.

SAME?

One block of code is compared with another.Only 3 of the 4 block enders need to be legal; the other is deduced. Any differences are reported.

DEFINE LABEL

A label plus its value is defined.

SHOW LABEL

The current labels are displayed.

```
/*****************************************************************/
/* STID:Windows, Main, Multi, Main Routines, Subsidaries:STID    */
/*****************************************************************/
/* INCLUDE FILES                                                 */
/*****************************************************************/
#include "stdio.h"
#include "obdefs.h"
#include "define.h"
#include "gemdefs.h"
#include "osbind.h"
#include "stidmenu.h"
/*****************************************************************/
/* DEFINES                                                       */
/*****************************************************************/
#define WI_KIND    (SIZER|MOVER|FULLER|CLOSER|NAME)
#define WI_KIND2   (SIZER|MOVER|FULLER|CLOSER|NAME|UPARROW|DNARROW|VSLIDE)

#define LAB_LENGTH (8)
#define MAXLABS    (50)
#define MAXFNS     (5)
#define MIN_WIDTH  (2*gl_wbox)
#define MIN_HEIGHT (2*gl_hbox)

#define TOPMEM    (0xFFFFL)        /* for 1040 STF */
#define ELLIPSE   (0)
#define MEMORY    (1)
#define LABELS    (2)
#define OUTPUT    (3)
#define FIND      (4)
#define REPLACE   (5)
#define FILL      (6)
#define COPY      (7)
#define SAME      (8)
#define BREAKS    (9)
#define RARROW    (77)
#define LARROW    (75)
#define UARROW    (72)
#define DARROW    (80)
/*****************************************************************/
/* EXTERNALS                                                     */
/*****************************************************************/
extern int      gl_apid;
/*****************************************************************/
/* GLOBAL VARIABLES                                              */
/*****************************************************************/
typedef int (*pt_to_fn)();

pt_to_fn last_command;
struct window
  {int xwork,ywork,hwork,wwork,xold,yold,hold,wold,on,type,han,fulled,bwl,tag;
```

```
    form_alert();
      long loc;} wlist;    /* name=handle,loc=location or first mem location in
                              window */
    struct lab_item
           {char label[LAB_LENGTH];long value;} labels[MAXLABS];

    struct lab_item templab;

    struct fn_key
           {int active,num_fns;pt_to_fn fns[MAXFNS];} fn_keys[10];
    /* For the function keys to work, each facility in the menu must return
       an integer */
    int    open_awindow(),inspect(),open_lwindow(),find(),replace(),fill(),
           copy(),same(),def_lab(),load_hex();

    char   fs_iinpath[]="A:/*.RSC";
    char   fs_iinsel[]="STIDMENU.RSC";
    int    fs_exit;

    long   hilite;
    int    hilite_on=FALSE;

    int    brk_flag=FALSE;
    int    lab_flag=FALSE;
    int    exit_flag=FALSE;

    long   lblock=0xffe80,hblock=0xfffa0,lblock1,hblock1,lblock2,hblock2,bl_size;
    long   low_limit,high_limit,length,bytes_visible;
    int    button;
    char   * addr1;
    char   * addr;
    int    bytes_per_line=8;                  /* default for memory display */
    int    po_char;                           /* used in locator and cursor */
    int    xchar,ychar,xhex,yhex;      /* x,y coords for memory changing  */

    int    tag=0;                           /* the number of tagged windows */
    int    redef=0,newkey,fn_key_hit;              /* for key redefinition */
    int    gl_hchar,gl_wchar,gl_wbox,gl_hbox;              /* system sizes */
    int    mask=255;
    int    phys_handle;                      /* physical workstation handle */
    int    handle;                           /* virtual workstation handle */

    int    windcount =0;
    int    maxnumwinds=5;        /* leaving one free for output etc        */
    int    top_handle;          /* = w[stack[0]].han ,handle of top window */
    int    topwnum;              /* = stack[0], struc index of top window */
    int    stack[5];           /* not a true stack.It holds the depth-order
                                  of the windows                          */
    int    xdesk,ydesk,hdesk,wdesk;
    char   target[80];
    char   string[80];
    char   wild[80];
    char   chbuff[80];
    char   numbuff[80];
    int    msgbuff[8];       /* event message buffer */
    int    keycode;          /* keycode returned by event-keyboard */
    int    num_clicks;
    int    mx,my;            /* mouse x and y pos. */
    int    butdown;          /* button state tested for, UP/DOWN */
    int    ret;              /* dummy return variable */
    long   total;
    int    i,j,k;            /* to be used with care */
    char   ch;

    int    no_of_labs=7;    /* For test purposes I've initialised some labels */
    int    first_label=0;
    int    picked_label;
```

```
long    menu_tree;
int     temp[4];

int     contrl[12];
int     intin[128];
int     ptsin[128];
int     intout[128];
int     ptsout[128];

int work_in[11];        /* Input to GSX parameter array */
int work_out[57];       /* Output from GSX parameter array */
int pxyarray[10];       /* input point array */


char tit1[]= "Find";
char tit2[]= "Replace";
char tit3[]= "Inspect";
char tit4[]= "Fill Block";
char tit5[]= "Copy Block";
char tit6[]= "Same?";
char tit7[]= "Define label";

char te_to[]= "to:-_____";
char te_fr[]= "from:-_____";
char te_str[]= "string:-_____";
char te_byt[]= "bytes:-__/__/__";
char te_sub[]= "new string:-_____";
char te_lab[]= "label:-_____";
char te_val[]= "value:-_____";

char te_toval[]= "Xnnnnn";
char te_frval[]= "Xnnnnn";
char te_strval[]= "XXXXXX";
char te_bytval[]= "nnnnnn";
char te_subval[]= "XXXXXX";
char te_labval[]= "XXXXXXXX";
char te_valval[]= "nnnnnn";

char te_totext[]= "_____";
char te_frtext[]= "_____";
char te_toltext[]= "_____";
char te_fritext[]= "_____";
char te_strtext[]= "_____";
char te_byttext[]= "_____";
char te_subtext[]= "_____";
char te_labtext[]= "_____";

char te_ok[]="ok";
char te_can[]="cancel";
char te_byte[]="byte";
char te_word[]="word";
char te_long[]="long";

TEDINFO ti_to =(te_totext,te_to,te_toval,IBM,0,TE_CNTR,0x11F0,0,1,7,12};
TEDINFO ti_fr =(te_frtext,te_fr,te_frval,IBM,0,TE_CNTR,0x11F0,0,1,7,14};
TEDINFO ti_to1 =(te_toltext,te_to,te_toval,IBM,0,TE_CNTR,0x11F0,0,1,7,12};
TEDINFO ti_fr1 =(te_fritext,te_fr,te_frval,IBM,0,TE_CNTR,0x11F0,0,1,7,14};
TEDINFO ti_str =(te_strtext,te_str,te_strval,IBM,0,TE_CNTR,0x11F0,0,1,7,16};
TEDINFO ti_byt =(te_byttext,te_byt,te_bytval,IBM,0,TE_CNTR,0x11F0,0,1,7,17};
TEDINFO ti_strsub =(te_subtext,te_sub,te_subval,IBM,0,TE_CNTR,0x11F0,0,1,7,20};
TEDINFO ti_lab =(te_labtext,te_lab,te_labval,IBM,0,TE_CNTR,0x11F0,0,1,9,16};
TEDINFO ti_val =(te_subtext,te_val,te_valval,IBM,0,TE_CNTR,0x11F0,0,1,7,14};

/*-------------------------------------------------------------------------*/
OBJECT o_find= {/* 0 */ -1,1,6,G_BOX,NONE,NORMAL,0x00021160L,5,5,350,140};
OBJECT o_tit1= {/* 1 */ 2,-1,-1,G_BUTTON,NONE,NORMAL,0x1100,5,5,200,20};
```

```
OBJECT o_ctl1=  {/* 1 */ 2,-1,-1,G_BUTTON,NONE,NORMAL,tit1,50,5,200,20};
OBJECT o_fr=    {/* 2 */ 3,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_fr,20,30,130,20};
OBJECT o_to =   {/* 3 */ 4,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_to,20,55,130,20};
OBJECT o_str=   {/* 4 */ 5,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_str,
                                                        20,80,130,20};
OBJECT o_ok=    {/* 5 */ 6,-1,-1,G_BUTTON,SELECTABLE|EXIT|DEFAULT,NORMAL,te_ok,
                                                        20,105,64,24};
OBJECT o_canc=  {/* 6 */ 0,-1,-1,G_BUTTON,SELECTABLE|EXIT|LASTOB,NORMAL,te_can,
                                                        150,105,64,24};
/*------------------------------------------------------------------------*/
OBJECT o_replace={/* 0 */ -1,1,7,G_BOX,NONE,NORMAL,0x00021160L,5,5,300,170};
OBJECT o_tit2=    {/* 1 */ 2,-1,-1,G_BUTTON,NONE,NORMAL,tit2,50,5,200,20};
OBJECT o_fr1=     {/* 2 */ 3,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_fr,
                                                        20,30,130,20};
OBJECT o_to1 =    {/* 3 */ 4,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_to,
                                                        20,55,130,20};
OBJECT o_str1=    {/* 4 */ 5,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_str,
                                                        20,80,130,20};
OBJECT o_sub=     {/* 5 */ 6,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_strsub,
                                                        20,105,160,20};
OBJECT o_ok1=     {/* 6 */ 7,-1,-1,G_BUTTON,SELECTABLE|EXIT|DEFAULT,NORMAL,te_ok,
                                                        20,130,64,24};
OBJECT o_canc1=   {/* 7 */ 0,-1,-1,G_BUTTON,SELECTABLE|EXIT|LASTOB,NORMAL,te_can,
                                                        150,130,64,24};
/*------------------------------------------------------------------------*/
OBJECT o_inspect={/* 0 */ -1,1,7,G_BOX,NONE,NORMAL,0x00021160L,5,5,300,170};
OBJECT o_tit3=    {/* 1 */ 2,-1,-1,G_BUTTON,NONE,NORMAL,tit3,50,5,200,20};
OBJECT o_fr2=     {/* 2 */ 3,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_fr,
                                                        20,40,130,20};
OBJECT o_byte=    {/* 3 */ 4,-1,-1,G_BUTTON,SELECTABLE|RBUTTON,NORMAL,te_byte,
                                                        20,80,65,24};
OBJECT o_word=    {/* 4 */ 5,-1,-1,G_BUTTON,SELECTABLE|RBUTTON,SELECTED,te_word,
                                                        86,80,65,24};
OBJECT o_long=    {/* 5 */ 6,-1,-1,G_BUTTON,SELECTABLE|RBUTTON,NORMAL,te_long,
                                                        152,80,65,24};
OBJECT o_ok2=     {/* 6 */ 7,-1,-1,G_BUTTON,SELECTABLE|EXIT|DEFAULT,NORMAL,te_ok,
                                                        20,130,65,24};
OBJECT o_canc2=   {/* 7 */ 0,-1,-1,G_BUTTON,SELECTABLE|EXIT|LASTOB,NORMAL,te_can,
                                                        152,130,65,24};
/*------------------------------------------------------------------------*/
OBJECT o_fill=  {/* 0 */ -1,1,6,G_BOX,NONE,NORMAL,0x00021160L,5,5,300,190};
OBJECT o_tit4=  {/* 1 */ 2,-1,-1,G_BUTTON,NONE,NORMAL,tit4,50,5,200,20};
OBJECT o_fr3=   {/* 2 */ 3,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_fr,
                                                        20,30,130,24};
OBJECT o_to3 =  {/* 3 */ 4,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_to,
                                                        20,55,130,24};
OBJECT o_sub3=  {/* 4 */ 5,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_strsub,
                                                        20,105,160,24};
OBJECT o_ok3=   {/* 5 */ 6,-1,-1,G_BUTTON,SELECTABLE|EXIT|DEFAULT,NORMAL,te_ok,
                                                        20,130,64,24};
OBJECT o_canc3={/* 6 */ 0,-1,-1,G_BUTTON,SELECTABLE|EXIT|LASTOB,NORMAL,te_can,
                                                        20,155,64,24};
/*------------------------------------------------------------------------*/
OBJECT o_copy=  {/* 0 */ -1,1,7,G_BOX,NONE,NORMAL,0x00021160L,5,5,300,190};
OBJECT o_tit5=  {/* 1 */ 2,-1,-1,G_BUTTON,NONE,NORMAL,tit5,50,5,200,20};
OBJECT o_sfr=   {/* 2 */ 3,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_fr,
                                                        20,30,130,24};
OBJECT o_sto =  {/* 3 */ 4,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_to,
                                                        20,55,130,24};
OBJECT o_dfr=   {/* 4 */ 5,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_fr1,
                                                        160,30,130,24};
OBJECT o_dto =  {/* 5 */ 6,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_to1,
                                                        160,55,130,24};
OBJECT o_ok4=   {/* 6 */ 7,-1,-1,G_BUTTON,SELECTABLE|EXIT|DEFAULT,NORMAL,te_ok,
                                                        20,130,64,24};
OBJECT o_canc4={/* 7 */ 0,-1,-1,G_BUTTON,SELECTABLE|EXIT|LASTOB,NORMAL,te_can,
```

```
/*-------------------------------------------------------------------------*/
OBJECT o_same= {/* 0 */ -1,1,7,G_BOX,NONE,NORMAL,0x00021160L,5,5,300,120};
OBJECT o_tit6= {/* 1 */ 2,-1,-1,G_BUTTON,NONE,NORMAL,tit6,50,5,200,20};
OBJECT o_sfr1= {/* 2 */ 3,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_fr,
                                                        20,30,130,24};
OBJECT o_sto1 ={/* 3 */ 4,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_to,
                                                        20,55,130,24};
OBJECT o_dfr1= {/* 4 */ 5,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_fr1,
                                                        160,30,130,24};
OBJECT o_dto1 ={/* 5 */ 6,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_to1,
                                                        160,55,130,24};
OBJECT o_ok5=  {/* 6 */ 7,-1,-1,G_BUTTON,SELECTABLE|EXIT|DEFAULT,NORMAL,te_ok,
                                                        20,85,64,24};
OBJECT o_canc5={/* 7 */ 0,-1,-1,G_BUTTON,SELECTABLE|EXIT|LASTOB,NORMAL,te_can,
                                                        130,85,64,24};
/*-------------------------------------------------------------------------*/
OBJECT o_lab=  {/* 0 */ -1,1,5,G_BOX,NONE,NORMAL,0x00021160L,5,5,300,190};
OBJECT o_tit7= {/* 1 */ 2,-1,-1,G_BUTTON,NONE,NORMAL,tit7,50,5,200,20};
OBJECT o_lab1= {/* 2 */ 3,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_lab,
                                                        20,30,130,24};
OBJECT o_val = {/* 3 */ 4,-1,-1,G_FBOXTEXT,EDITABLE,NORMAL,&ti_val,
                                                        20,55,130,24};
OBJECT o_ok6=  {/* 4 */ 5,-1,-1,G_BUTTON,SELECTABLE|EXIT|DEFAULT,NORMAL,te_ok,
                                                        20,130,64,24};
OBJECT o_canc6={/* 5 */ 0,-1,-1,G_BUTTON,SELECTABLE|EXIT|LASTOB,NORMAL,te_can,
                                                        20,155,64,24};
/*-------------------------------------------------------------------------*/
/*****************************************************************/
/*  mouse routines.                                              */
/*****************************************************************/
hide_mouse()
{v_hide_c(handle);}

show_mouse()
{ v_show_c(handle,1);}
/*****************************************************************/
/* open virtual workstation                                      */
/*****************************************************************/
open_vwork()
{int i;
 for(i=0;i<10;work_in[i++]=1);
 work_in[10]=2;
 handle=phys_handle;
 v_opnvwk(work_in,&handle,work_out);
}
/*****************************************************************/
/* set clipping rectangle                                        */
/*****************************************************************/
set_clip(x,y,w,h)
int x,y,w,h;
{int clip[4];
 clip[0]=x;
 clip[1]=y;
 clip[2]=x+w;
 clip[3]=y+h;
 vs_clip(handle,1,clip);
}
/*****************************************************************/
/* create window for OUTPUT.  Parameter is struct index          */
/*****************************************************************/
open_owindow(vacant)
int vacant;
{w[vacant].type=OUTPUT;
 w[vacant].han=wind_create(WI_KIND,xdesk,ydesk,wdesk,hdesk);
 wind_set(w[vacant].han, WF_NAME," OUTPUT ",0,0);
 for(i=windepth-1;i>1;i--)  /*for all windows deepest first        */
```

```
 openw(vacant);
 cls(vacant);
}
/*************************************************************/
/* create window for ellipse.                                */
/*************************************************************/
open_ewindow()
{int vacant,height;
 vacant=find_vacant();if(vacant(0) return(0);
 w[vacant].type=ELLIPSE;
 w[vacant].han=wind_create(WI_KIND,xdesk,ydesk,wdesk,hdesk);
 wind_set(w[vacant].han, WF_NAME," ELLIPSE ",0,0);
 openw(vacant);
 draw_ellipse(vacant);
 return(1);
}
/*************************************************************/
/* create window for memory.                                 */
/*************************************************************/
open_mwindow(start,bwl)
int bwl;long start;
{int vacant;
 vacant=find_vacant();if(vacant(0){wind_alert();return;}
 w[vacant].bwl=bwl;w[vacant].loc=bwl*(start/bwl);
 w[vacant].type=MEMORY;
 w[vacant].han=wind_create(WI_KIND2,xdesk,ydesk,wdesk,hdesk);
 openw(vacant);
 showmem(topwnum);
}
/*************************************************************/
/****** create window for labels                        ******/
/*************************************************************/
open_lwindow()
{int vacant;
 if(lab_flag){en_lab_alert();return(0);}
 lab_flag=TRUE;              /* so there can't be 2 of these windows */
 vacant=find_vacant();if(vacant(0){wind_alert();return(0);}
 w[vacant].type=LABELS;
 w[vacant].han=wind_create(WI_KIND2,xdesk,ydesk,wdesk,hdesk);
 wind_set(w[vacant].han, WF_NAME," labels  ",0,0);
 openw(vacant);
 showlabs(vacant);
 return(1);
}
/*************************************************************/
/****** open  top window(1st on left half of screen, 2nd on right etc)*/
/*************************************************************/
openw(vacant)
int vacant;
{int height,width;
 if(w[vacant].type==LABELS)
   {height= min(hdesk,(3+no_of_labs)*gl_hchar);width=28*gl_wchar;}
 else {height=hdesk;width=wdesk/2;}
 w[vacant].on=TRUE;
 wind_open(w[vacant].han,xdesk+(vacant%2)*wdesk/2,ydesk,width,height);
 wind_get(w[vacant].han,WF_WORKXYWH,
          &w[vacant].xwork,&w[vacant].ywork,&w[vacant].wwork,&w[vacant].hwork);
 top_handle=w[vacant].han;topwnum=vacant;
 newwind(vacant);
 windcount++;
}
/*************************************************************/
/*              clear window                                 */
/*************************************************************/
cls(wnum)
int wnum;
{wf_interior(handle,2);
```

```
 vsf_style(handle,8);
 vsf_color(handle,0);
 wind_get(w[wnum].han,WF_WORKXYWH,temp,temp+1,temp+2,temp+3);
 temp[2]+=temp[0]-1;
 temp[3]+=temp[1]-1;
 v_bar(handle,temp);
}
/*****************************************************************/
/****** close top window                                        */
/*****************************************************************/
close_wi()
{         int i;
          w[topwnum].on=FALSE;
          switch(w[topwnum].type)
                {case LABELS: lab_flag=FALSE;hilite_on=FALSE;break;
                      /* but hilited addr may not be here! */
                 case BREAKS: brk_flag=FALSE;break;
                 case MEMORY: if(w[topwnum].tag)
                                {w[topwnum].tag=FALSE;
                                 tag--;
        /* if this leaves only one window tagged then detag the window */
                                 if(tag==1){for(i=0;i<4;i++)
                                                 w[i].tag=FALSE;
                                            tag=0;
                                           }
                                }
                }
          wind_close(w[topwnum].han);
          wind_delete(w[topwnum].han);
          windcount--;
          close_top_window();
          topwnum=stack[0];top_handle=w[topwnum].han;
}       /* note that closing topwindow auto-tops the window beneath
           so that topwnum, etc will not be updated in multi()       */
/**********************************************************************/
/*               open dialog box                                      */
/************************************************************************/
int dialbox(o_dial,first_field)
long o_dial;int first_field;
{int x,y,w,h,ex_but;
form_center(o_dial,&x,&y,&w,&h);
form_dial(0,10,10,10,10,x,y,w,h);
form_dial(1,10,10,10,10,x,y,w,h);
hide_mouse();
objc_draw(o_dial,0,10,x,y,w,h);
show_mouse();
ex_but=form_do(o_dial,first_field);
form_dial(2,10,10,10,10,x,y,w,h);
form_dial(3,10,10,10,10,x,y,w,h);
return(ex_but);
}
/*****************************************************************/
/* Find and redraw all clipping rectangles                       */
/*****************************************************************/
/* This doesn't work cleanly. Reason unknown*/
do_redraw(xc,yc,wc,hc)
int xc,yc,wc,hc;
{GRECT t1,t2;
 int i,the_window;
 hide_mouse();
 wind_update(TRUE);
 t2.g_x=xc;
 t2.g_y=yc;
 t2.g_w=wc;
 t2.g_h=hc;
```

```
for(i=num_windows-1;i>=0;i--) /* for all windows,deepest first */
    { the_window=stack[i];
      wind_get(w[the_window].han,WF_FIRSTXYWH,&t1.g_x,&t1.g_y,&t1.g_w,&t1.g_h);
      while (t1.g_w && t1.g_h)
            {if (rc_intersect(&t2,&t1))
                {set_clip(t1.g_x,t1.g_y,t1.g_w,t1.g_h);
                 switch (w[the_window].type)
                        {case ELLIPSE:draw_ellipse(the_window);break;
                         case MEMORY: showmem(the_window);break;
                         case LABELS: showlabs(the_window);break;
                        }
                }
             wind_get(w[the_window].han,WF_NEXTXYWH,&t1.g_x,&t1.g_y,&t1.g_w,&t1.
g_h);
            }/* end while */
    }/* end for */
        wind_update(FALSE);
        show_mouse();
}
/*****************************************************************/
/*              Accessory Init. Until First Event_Multi          */
/*****************************************************************/
main()
{
        appl_init();
        phys_handle=graf_handle(&gl_wchar,&gl_hchar,&gl_wbox,&gl_hbox);
        wind_get(0, WF_WORKXYWH, &xdesk, &ydesk, &wdesk, &hdesk);
        open_vwork();
        graf_mouse(ARROW,0x0L);
        buttdown=TRUE;
        if(!rsrc_load("stidmenu.rsc"))
          {form_alert(1,"[3][no resources][sorry!]");appl_exit();}
        if(rsrc_gaddr(0,0,&menu_tree)==0)
          {form_alert(1,"[3][ABORT][sorry!]");appl_exit();}

        menu_bar(menu_tree,1);

        last_command=inspect;  /* say, */
        for(i=0;i<4;i++)
            {w[i].en=0;w[i].fulled=FALSE;}
        for(i=0;i<11;i++)
            fn_keys[i].active=FALSE;
        for(i=0;i<LAB_LENGTH;i++)
            string[i]=255;
            string[i]=0;
        strcpy(labels[0].label,"label");labels[0].value=0xfff10;
        strcpy(labels[1].label,"label1");labels[1].value=0xfff30;
        strcpy(labels[2].label,"label2");labels[2].value=0xfff50;
        strcpy(labels[3].label,"label3");labels[3].value=0xfff70;
        strcpy(labels[4].label,"label4");labels[4].value=0xfff80;
        strcpy(labels[5].label,"label5");labels[5].value=0xfff90;
        strcpy(labels[6].label,"label6");labels[6].value=0xfffa0;
        strcpy(labels[7].label,string);
        count_labels();
        multi();
}
/*****************************************************************/
/* dispatches all accessory tasks                                */
/*****************************************************************/
multi()
{
int event;

      do {
        num_clicks=0; /* necessary to avoid button-press and button-release
                         each being considered a button event.
                         Calc Chapter 6 shows the way to use this
```

```
                                        This program doesn't contain the correct trick */
        event = evnt_multi(MU_MESAG | MU_BUTTON | MU_KEYBD,
                           2,1,buttdown,
                           0,0,0,0,0,
                           0,0,0,0,0,
                           msgbuff,0,0,&mx,&my,&ret,&ret,&keycode,&num_clicks);

        wind_update(TRUE);

        if (event & MU_MESAG)
          switch (msgbuff[0]) {

          case MN_SELECTED:
                switch (msgbuff[4])
                {/*This section could be considerably shortened if each case
                   called a menu_selection procedure with the address of the
                   chosen function as parameter */
/*Go*/            case 21:if((redef==2)&&(fn_keys[newkey].num_fns(MAXFNS))
                              fn_keys[newkey].fns[(fn_keys[newkey].num_fns)++]=
                                 open_ewindow;
                            else  {last_command=open_ewindow;open_ewindow();}
                            break;
                case 27:if((redef==2)&&(fn_keys[newkey].num_fns(MAXFNS))
                              fn_keys[newkey].fns[(fn_keys[newkey].num_fns)++]=
                                 inspect;
                            else  {last_command=inspect;inspect();}
                            break;
                case 44:if((redef==2)&&(fn_keys[newkey].num_fns(MAXFNS))
                              fn_keys[newkey].fns[(fn_keys[newkey].num_fns)++]=
                                 open_lwindow;
                            else  {last_command=open_lwindow;open_lwindow();}
                            break;
                case 30:if((redef==2)&&(fn_keys[newkey].num_fns(MAXFNS))
                              fn_keys[newkey].fns[(fn_keys[newkey].num_fns)++]=
                                 find;
                            else  {last_command=find;find();}
                            break;
                case 31:if((redef==2)&&(fn_keys[newkey].num_fns(MAXFNS))
                              fn_keys[newkey].fns[(fn_keys[newkey].num_fns)++]=
                                 replace;
                            else  {last_command=replace;replace();}
                            break;
                case 34:if((redef==2)&&(fn_keys[newkey].num_fns(MAXFNS))
                              fn_keys[newkey].fns[(fn_keys[newkey].num_fns)++]=
                                 fill;
                            else  {last_command=fill;fill();}
                            break;
                case 40:if((redef==2)&&(fn_keys[newkey].num_fns(MAXFNS))
                              fn_keys[newkey].fns[(fn_keys[newkey].num_fns)++]=
                                 copy;
                            else  {last_command=copy;copy();}
                            break;
                case 28:if((redef==2)&&(fn_keys[newkey].num_fns(MAXFNS))
                              fn_keys[newkey].fns[(fn_keys[newkey].num_fns)++]=
                                 same;
                            else  {last_command=same;same();}
                            break;
                case 43:if((redef==2)&&(fn_keys[newkey].num_fns(MAXFNS))
                              fn_keys[newkey].fns[(fn_keys[newkey].num_fns)++]=
                                 def_lab;
                            else  {last_command=def_lab;def_lab();}
                            break;
                case 12:exit_flag=TRUE; /* exits whether key being defined
                                           or not */
                            break;
                case 51:if((redef==2)&&(fn_keys[newkey].num_fns(MAXFNS))
```

```
                                fn_keys[newkey].fns[(fn_keys[newkey].num_fns)++]=
                                        load_hex;
                                else  {last_command=load_hex;load_hex();}
                                break;
                        default:break;
                        }

                menu_tnormal(menu_tree,msgbuff[3],1);  /* restore normal type */
                break;

        case WM_REDRAW:
                do_redraw(msgbuff[4],msgbuff[5],msgbuff[6],msgbuff[7]);
                break;

        case WM_ARROWED:
                if ((msgbuff[4]==2)&&(w[topwnum].type==MEMORY))
                   {if(tag){for(i=0;i<4;i++)
                             if(w[i].tag)w[i].loc-=bytes_per_line;}
                    else w[topwnum].loc-=bytes_per_line;
                   }
                if ((msgbuff[4]==3)&&(w[topwnum].type==MEMORY))
                   {if(tag){for(i=0;i<4;i++)
                             if(w[i].tag)w[i].loc+=bytes_per_line;}
                    else w[topwnum].loc+=bytes_per_line;
                   }
                if (w[topwnum].type==LABELS)
                   { if(msgbuff[4]==2)first_label=max(0,first_label-1);
                     if(msgbuff[4]==3)first_label=min(no_of_labs-1,first_label+1);
                   }
                if ((tag)&&(w[topwnum].type==MEMORY)) /* then refresh mem winds */
                    {for(i=windcount-1;i>-1;i--)
                          if (w[stack[i]].tag)
                                {wind_get(w[stack[i]].han,WF_WORKXYWH,
                                  temp,temp+1,temp+2,temp+3);
                                 do_redraw(temp[0],temp[1],temp[2],temp[3]);
                                }
                    }
                else{wind_get(w[topwnum].han,WF_WORKXYWH,
                                temp,temp+1,temp+2,temp+3);
                     do_redraw(temp[0],temp[1],temp[2],temp[3]);
                    }
                break;

        case WM_VSLID:
                switch (w[topwnum].type)
                {case MEMORY: /* This crashes */
                      bytes_visible=(w[topwnum].hwork*bytes_per_line)/gl_hchar;
                      if((w[topwnum].loc<lblock)||
                        ((w[topwnum].loc+bytes_visible)>hblock))
                          {low_limit=0x0L;  high_limit=TOPMEM;}
                      else  {low_limit=lblock;high_limit=hblock;}
                      length=high_limit-low_limit;
                      w[topwnum].loc=low_limit+(msgbuff[4])*length/1000;
                      wind_set(w[topwnum].han,WF_VSLIDE,msgbuff[4],0,0,0);
/*???*/                 printf("%ld.",(msgbuff[4])*(length)/1000);
                      break;
                 case LABELS:
                      first_label=max(0,msgbuff[4]*(no_of_labs-1)/1000);
                      wind_set(w[topwnum].han,WF_VSLIDE,msgbuff[4],0,0,0);
                 } /*switch type end */
                wind_get(w[topwnum].han,WF_WORKXYWH,temp,temp+1,temp+2,temp+3);
                do_redraw(temp[0],temp[1],temp[2],temp[3]);
                break;

        case WM_NEWTOP:
        case WM_TOPPED:
```

```
            wind_set(msgbuff[3],WF_TOP,0,0,0,0);
            top_handle=msgbuff[3];
            for (i=0;i<=maxnumwinds;i++)
                {if (w[i].han==top_handle){topwnum=i;break;} }
            topper(topwnum);
            break;

          case WM_SIZED:
          case WM_MOVED:
            if(msgbuff[6]<MIN_WIDTH)msgbuff[6]=MIN_WIDTH;
            if(msgbuff[7]<MIN_HEIGHT)msgbuff[7]=MIN_HEIGHT;
            wind_set(top_handle,WF_CURRXYWH,
                     msgbuff[4],msgbuff[5],msgbuff[6],msgbuff[7]);
            wind_get(top_handle,WF_WORKXYWH,
            &w[topwnum].xwork,&w[topwnum].ywork,
            &w[topwnum].wwork,&w[topwnum].hwork);
            break;

          case WM_FULLED:    /* this looks too long */
            if(w[topwnum].type==MEMORY)ret= WI_KIND;else ret=WI_KIND2;
            if(w[topwnum].fulled)
              {wind_calc(WC_WORK,ret,
                w[topwnum].xold,w[topwnum].yold,w[topwnum].wold,w[topwnum].hold,
                &w[topwnum].xwork,&w[topwnum].ywork,
                &w[topwnum].wwork,&w[topwnum].hwork);
               wind_set(w[topwnum].han,WF_CURRXYWH,
                w[topwnum].xold,w[topwnum].yold,
                w[topwnum].wold,w[topwnum].hold);
              }
            else{wind_calc(WC_BORDER,ret,
                 w[topwnum].xwork,w[topwnum].ywork,
                 w[topwnum].wwork,w[topwnum].hwork,
                 &w[topwnum].xold,&w[topwnum].yold,
                 &w[topwnum].wold,&w[topwnum].hold);
                wind_calc(WC_WORK,ret,xdesk,ydesk,wdesk,hdesk,
                 &w[topwnum].xwork,&w[topwnum].ywork,
                 &w[topwnum].wwork,&w[topwnum].hwork);
                wind_set(w[topwnum].han,WF_CURRXYWH,xdesk,ydesk,wdesk,hdesk);
                }
            w[topwnum].fulled ^= TRUE;
            break;

          case WM_CLOSED:
            close_wi();
            break;
          } /* switch (msgbuff[0]) end */

            /* now check for tagging. Presently triggered by F8 */
            /* which tags all operational memory windows        */
        if((event & MU_KEYBD)&&((keycode/256)==66)
            &&(w[topwnum].type==MEMORY))
           {tag=0;
            for(i=0;i<maxnumwinds;i++)
                 if((w[i].type==MEMORY)&&(w[i].on==TRUE)){w[i].tag=TRUE;tag++;}
            if(tag==1)
               {for(i=0;i<maxnumwinds;i++)
                    w[i].tag=0;
                tag=0;
               }
           }
            /* check for 'repeat last command' .Key F7          */
        if((event & MU_KEYBD)&&((keycode/256)==65))
          (*last_command)();

/*****************************************************************/
/* wind_color(town);                                             */
```

```
    /* check fn keys.                                                    */
    /*    redef=0       Normal operation                                 */
    /*    redef=1       F1 has been pressed. Await another fn key press. */
    /*    redef=2       Await redefinition of newkey                     */
    /*********************************************************************/
                    if((event & MU_KEYBD)&&(keycode/256 >58)&&
                      (keycode/256 (69))
                     {fn_key_hit= keycode/256 -58;
                      if((fn_key_hit==1)&&(!redef))redef=1;
                      if((fn_key_hit)1)&&(fn_key_hit(10)
                          &&(!redef)&&(fn_keys[fn_key_hit].active))
                          {i=0;
                           while(i ( fn_keys[fn_key_hit].num_fns)
                                (*fn_keys[fn_key_hit].fns[i++])();
                          }
                      if((fn_key_hit)1)&&(fn_key_hit(6)&&(redef==1))
                          {redef++;newkey=fn_key_hit;fn_keys[newkey].num_fns=0;
                          }
                      if((fn_key_hit==10)&&(redef==2))
                          {redef=0;fn_keys[newkey].active=TRUE;
                          }
                     }
    /* check for button event in window */
             if ((event & MU_BUTTON) && (w[topwnum].han==wind_find(mx,my))
                  &&(num_clicks)0))
              switch (w[topwnum].type)
                      {case MEMORY:
                                  locator();
                                  butdown ^=TRUE;
                                  break;
                       case LABELS:
                                  picked_label=(my-w[topwnum].ywork)/gl_hchar+
                                                first_label;
                                  if (picked_label)=no_of_labs)break;
                                  if(hilite_on) /* then erase old visible hilite */
                                    {for(i=0;i(no_of_labs;i++)
                                      if((labels[i].value==hilite)&&
                                          (i)=first_label)&&
                                          (i(min(no_of_labs,
                                           first_label+w[topwnum].hwork/gl_hchar)))
                                           {temp[0]= w[topwnum].xwork;
                                            temp[1]= w[topwnum].ywork+
                                                     gl_hchar*(1+i-first_label);
                                            temp[2]= temp[0]+w[topwnum].wwork-1;
                                            temp[3]= temp[1]-gl_hchar;
                                            inv_color(temp);
                                           }
                                     }
                                  hilite=labels[picked_label].value;hilite_on=TRUE;
                                  temp[0]=w[topwnum].xwork;
                                  temp[1]=w[topwnum].ywork+
                                          gl_hchar*(1+picked_label-firstk_label);
                                  temp[2]=temp[0]+w[topwnum].wwork-1;
                                  temp[3]=temp[1]-gl_hchar;
                                  inv_color(temp);
                       butdown ^=TRUE;
                       break;
                       } /* endswitch on type */
              wind_update(FALSE);

            }while(!exit_flag);

            while(windcount)0)
                  { close_wi();}
            menu_bar(menu_tree,0);
            v_clsvwk(handle);
                                  /*EXIT FROM PROGRAM HERE*/
```

```
    appl_exit();          /*EXIT PROGRAM HERE*/
}
/*************************************************************************/
/* Draw Filled Ellipse. Retained for test purposes to check re_draw */
/*************************************************************************/
draw_ellipse(wnum)
int wnum;
{cls(wnum);
 vsf_interior(handle,4);
 vsf_color(handle,1);
 v_ellipse(handle,w[wnum].xwork+w[wnum].wwork/2,
 w[wnum].ywork+w[wnum].hwork/2,w[wnum].wwork/2,w[wnum].hwork/2);
}
/*************************************************************************/
/* show memory                                                           */
/*************************************************************************/
 showmem(wnum)
 int wnum;
{char * bpointer;
 int  * wpointer;
 long * lpointer;
 long bytes_visible;
 long numerator,num2erator,length;
 long low_limit,high_limit;
 int i,j=gl_hchar,k;
 int bwl=w[wnum].bwl;
 cls(wnum);
 bytes_visible=(w[wnum].hwork*bytes_per_line)/gl_hchar;
 if((w[wnum].loc+bblock)!!((w[wnum].loc+bytes_visible))hblock))
   {low_limit=0x0L;high_limit=TOPMEM;
    wind_set(w[wnum].han,WF_NAME,"main memory",0,0);
   }
 else {low_limit=bblock;high_limit=hblock;
       wind_set(w[wnum].han,WF_NAME,"block memory",0,0);
      }
 length=high_limit-low_limit;
 numerator=1000*(w[wnum].loc - low_limit);
 num2erator=1000*bytes_visible;
 i=numerator/length;
 k=num2erator/length;
 wind_set(w[wnum].han,WF_VSLIDE,
          i,0,0,0);
 wind_set(w[wnum].han,WF_VSLSIZE,
          k,0,0,0);
 wind_get(w[wnum].han,WF_WORKXYWH,temp,temp+1,temp+2,temp+3);
 set_clip(temp[0],temp[1],temp[2]-1,temp[3]-1);
 for(i=0;i<bytes_visible;i+=bytes_per_line)
    {bpointer=w[wnum].loc+i;wpointer=bpointer;lpointer=bpointer;
     for (k=0;k<bytes_per_line;k++)
           {ch= *(bpointer+k)&mask;
            if(bwl==1)sprintf(numbuff+k*3,"%02x.",ch);
            if((bwl==2)&&((k/2)*2==k))sprintf(numbuff+(k/2)*5,"%04x.",
                  *(wpointer+k/2));
            if((bwl==4)&&((k/4)*4==k))sprintf(numbuff+(k/4)*9,"%08Lx.",
                  *(lpointer+k/4));
            if( (ch<040)!!(ch>0176) )ch='_';/* unprintable */
            chbuff[k]=ch;
           }
     chbuff[k]=0;   /* terminate string with 0 */
     sprintf(string,"%06Lx__%s%s",bpointer,numbuff,chbuff);
     v_gtext(handle,w[wnum].xwork+10,w[wnum].ywork+j,string);
     if((bpointer==hilite)&&(hilite_on))
        {temp[0]=w[wnum].xwork;
         temp[1]=w[wnum].ywork+j;
         temp[2]=temp[0]+10+6*gl_wchar;
         temp[3]=temp[1]-gl_hchar;
         inv_colm(temp);
```

```
/* check fn_keys;
        inv_color(temp);
        }
    j=j+gl_hchar;
    }
}
/*************************************************************************/
/* show labels                                                           */
/*************************************************************************/
int showlabs(wnum)
int wnum;
{int i,j=gl_hchar;
 long numerator=w[wnum].hwork*1000;
 cls(wnum);
 wind_set(w[wnum].han,WF_VSLIDE,
          max(1,(first_label*1000)/(1+no_of_labs)),0,0,0);
 i=numerator/(gl_hchar*(1+no_of_labs));
 wind_set(w[wnum].han,WF_VSLSIZE,max(1,i),0,0,0);
for (i=first_label;i<min(no_of_labs,first_label+w[wnum].hwork/gl_hchar);i++)
     {sprintf(string,"%12s----%06lx",labels[i].label,labels[i].value);
      v_gtext(handle,w[wnum].xwork+10,w[wnum].ywork+j,string);
      if((hilite_on)&&(labels[i].value==hilite))
        {temp[0]=w[wnum].xwork;
         temp[1]=w[wnum].ywork+gl_hchar*(i-clicked_label-first_label);
         temp[2]=temp[0]+w[wnum].wwork-1;
         temp[3]=temp[1]-gl_hchar;
         inv_color(temp);
        }
      j+=gl_hchar;
     }
return(1);
}
/*************************************************************************/
/*                              Find                                     */
/*************************************************************************/
find()
{int vacant,stleng,j,k;
 long from,to;
 sprintf(te_frtext,"%06lx",lblock);
 sprintf(te_totext,"%06lx",hblock);
 o_ok.ob_state=NORMAL;
 o_canc.ob_state=NORMAL;
 button=dialbox(&o_find,4);
 chop(te_srtext);
 stleng=strlen(te_srtext);
 if((button==5)&&(stleng))
    {if(!g_long(te_frtext)){num_alert();return(0);}else from=lotbl;
     if(!g_long(te_totext)){num_alert();return(0);}else to=hotbl;
     wildstr(te_srtext);
     vacant=find_vacant();if(vacant<0){wind_alert(); return(0);}
     hide_mouse();
     open_owindow(vacant);
     j=gl_hchar;
     while(findit(te_srtext,wild,from,to,&from))
           { for (k=0;k<bytes_per_line;k++)/* print a line of memory */
             {addr=from-2+k;
              ch= *(addr)&mask;
              sprintf(numbuff+k*3,"%02x.",ch);
              if( (ch<040)||(ch>0176) )ch='_';
              chbuff[k]=ch;
             }
              chbuff[k]=0;
              sprintf(string,"%06lx__%s%s",from,numbuff,chbuff);
              v_gtext(handle,w[topwnum].xwork+10,w[topwnum].ywork+j,string);
              j=j+gl_hchar;/* if j>w[topwnum].hwork SCROLL???? */
              from++;
              v_gtext(handle,w[topwnum].xwork+10,w[topwnum].ywork+j,
                      "press return to continue");
```

```
                while((mask & evnt_keybd())!=13);
              }
        close_wi();
        show_mouse();
        }
return(1);
}
/*****************************************************************************/
/*****              replace                                              ******/
/*****************************************************************************/
replace()
{int stleng;
 long low,high;
 sprintf(te_frtext,"%06Lx",lblock);
 sprintf(te_totext,"%06Lx",hblock);
 o_ok1.ob_state=NORMAL;
 o_canc1.ob_state=NORMAL;
 button=dialbox(&o_replace,5);
 if(button==6)
 {chop(te_subtext);chop(te_srtext);
  stleng=strlen(te_subtext);
  if((!stleng)||(stleng != strlen(te_srtext)))
    {form_alert(1,
     "[3][The length of the replacement ! doesn't match that of the ! target. Tr
y again.][ok]");
     return(0);
    }
  if(g_long(te_frtext)) low=total;else {num_alert();return(0);}
  if(g_long(te_totext)) high=total;else{num_alert();return(0);}
  wildstr(te_srtext);
  while(findit(te_srtext,wild,low,high,&low))
        for (k=0;k(stleng;k++)
            {addr=low+k;
             *addr=te_subtext[k];
            }
 }
return(1);
}
/*****************************************************************************/
/*          INSPECT                                                          */
/*****************************************************************************/
inspect()
{if (hilite_on)sprintf(te_frtext,"%06Lx",hilite);
    else sprintf(te_frtext,"%06Lx",lblock);
 o_ok2.ob_state=NORMAL;
 o_canc2.ob_state=NORMAL;
 button=dialbox(&o_inspect,2);
 if(button==6)
   {if((!g_long(te_frtext))){num_alert();return(0);}
    if(o_byte.ob_state & SELECTED) ret=1;
    if(o_word.ob_state & SELECTED) ret=2;
    if(o_long.ob_state & SELECTED) ret=4;
    open_mwindow(total,ret);
    return(1);
   }
return(0);
}
/*****************************************************************************/
/*          fill                                                             */
/*****************************************************************************/
fill()
{long low,high,ltemp;
 sprintf(te_frtext,"%06Lx",lblock);
 sprintf(te_totext,"%06Lx",hblock);
 o_ok3.ob_state=NORMAL;
```

```
    o_canc5.ob_state=NORMAL;
    button=dialbox(&o_fill,4);
    chop(te_subtext);
    if(button==5)
      {if(g_long(te_frtext)) low=total;else {num_alert();return(0);}
       if(g_long(te_totext)) high=total;else {num_alert();return(0);}
       ltemp=low;ret=strlen(te_subtext);
       if ((high-low)<=0) {num_alert();return(0);}
       if (high-low )>= ret)
          {for (ltemp=low;ltemp<high-ret;ltemp+=ret)
               for(j=0;j<ret;j++)
                   {addr=ltemp+j;addr1=te_subtext+j;
                    *addr=(*addr1);
                   }
          }
       for(j=0;j<=high-ltemp;j++)
          {addr=ltemp+j;addr1=te_subtext+j;
           *addr=(*addr1);
          }
      }
    return(1);
    }
    /*****************************************************************************/
    /*          copy                                                             */
    /*****************************************************************************/
    copy()
    {long i;
     sprintf(te_frtext,"%06Lx",lblock);
     sprintf(te_totext,"%06Lx",hblock);
     sprintf(te_fritext,"%06Lx",lblock);
     sprintf(te_toltext,"%06Lx",hblock);
     o_ok4.ob_state=NORMAL;
     o_canc4.ob_state=NORMAL;
     button=dialbox(&o_copy,2);
     if(button==6)
       {if(block2())
          if (lblock2<=lblock1)
               for(i=0;i<=bl_size;i++)
                   {addr=lblock1+i;addr1=lblock2+i;
                    *addr1=( *addr);
                   }
          else
               for(i=bl_size;i>=0;i--)
                   {addr=lblock1+i;addr1=lblock2+i;
                    *addr1=( *addr);
                   }
      }
    return(1);
    }
    /*****************************************************************************/
    /*          same                                                             */
    /*****************************************************************************/
    same()
    {int vacant,i,j;
     sprintf(te_frtext,"%06Lx",lblock);
     sprintf(te_totext,"%06Lx",hblock);
     sprintf(te_fritext,"%06Lx",lblock);
     sprintf(te_toltext,"%06Lx",hblock);
     o_ok5.ob_state=NORMAL;
     o_canc5.ob_state=NORMAL;
     button=dialbox(&o_same,2);
     if(button==6)
       {if(block2())
          {vacant=find_vacant();if(vacant<0){wind_alert(); return(0);}
           hide_mouse();
           open_owindow(vacant);
```

```
            for(i=0;i(b)_size;i++)
                {addr1=lblock1+i;addr=lblock2+i;
                 if (*addr1 != ( *addr))
                  {sprintf(string,"%06Lx_%02x...%06Lx__%02x",
                           addr,*addr & mask,addr1,*addr1 & mask );
                   v_gtext(handle,w[topwnum].xwork+10,w[topwnum].ywork+j,string);
                   j=j+gl_hchar;/* if j)w[topwnum].hwork the output will
                                   defemestrate. See prog notes        */
                   v_gtext(handle,w[topwnum].xwork+10,w[topwnum].ywork+j,
                           " press return to continue ");
                   while((mask & evnt_keybd())!=13);
                  }
                }
            close_wi();
            show_mouse();
           }
      }
return(1);
}
/*************************************************************************/
/*          define label                                                 */
/*************************************************************************/
def_lab()
{o_ok.ob_state=NORMAL;
 o_canc.ob_state=NORMAL;
 button=dialbox(&o_lab,2);
 if(button==4)
    {if (!g_long(te_subtext))return(0);
     chop(te_labtext);
     strcpy(templab.label,te_labtext);templab.value=total;
     add_label(&templab);
    }
return(1);
}
/*************************************************************************/
/*    load a new hex file                                                */
/*************************************************************************/
load_hex()
{fsel_input(fs_iinpath,fs_iinsel,&fs_exit);
                          /* Load Here and reset lblock,hblock*/
 return(1);
}
/*************************************************************************/
/*          window stack operations                                      */
/*************************************************************************/
close_top_window()
{int i;
 for( i=1;i(4;i++)
     stack[i-1]=stack[i];
}
/*************************************************************************/
topper(wnum)
int wnum;
{int i,j;
 if (wnum !=stack[0])
     for (i=1;i(windcount;i++)
          if (wnum==stack[i])
             { for (j=i;j)0;j--) stack[j]=stack[j-1];
               stack[0]=wnum;
               return;
             }
}
/*************************************************************************/
newwind(wnum)
int wnum;
{int i;
```

```
 for (i=3;i)0;i--)
       stack[i]=stack[i-1];
       stack[0]=wnum;
}
/*************************************************************************/
inv_color(temp) /* linker doesn't like this, so it's disabled */
int * temp;
{long ltemp=0x0L;
   hide_mouse();
   vro_cpyfm(handle,10,temp,&ltemp,&ltemp);
   show_mouse();
}
/*************************************************************************/
/* locator. deals with cursor press in memory window                     */
/*************************************************************************/
int locator()
{
char * addrl;
int key_hit;
int charstart;
int hexstart  =10+gl_wchar*8;   /* rel. x-coord of 1st hex */
int whichcol,whichblock,whichrow,inhex,lastccolvis,lasthcolvis;
int bwl,digits;
bwl=w[topwnum].bwl;
digits=bwl*2;
charstart= 10+gl_wchar*(8+((bytes_per_line)/bwl)*(digits+1));/* xcoord of 1st ch
ar */
po_char = mx-w[topwnum].xwork-charstart;
if (po_char)=0) /* then po a char */
     {
      whichcol= (po_char)/gl_wchar;
      whichblock=whichcol/bwl;
      inhex= 2*(whichcol%bwl);
      if (whichcol )=bytes_per_line  )return(0);  /*beyond the line end  */
     }
else  /* po a hex. Now calculate which b/w/l block is being pointed to */
     {whichblock= (mx-w[topwnum].xwork-hexstart)/((digits+1)*gl_wchar);
      if((inhex=((mx-w[topwnum].xwork-hexstart)%((digits+1)*gl_wchar))/gl_wchar)
                 )=digits ) return(0);      /*on an interhex space */
      whichcol= whichblock*bwl+inhex/2;
     }
whichrow=(my-w[topwnum].ywork)/gl_hchar;
if (whichblock(0)  /* before hex values, so hilite the address. Any hilite
                      already in this window should be de-hilighted.
                      (not implemented yet)*/
     {hilite=w[topwnum].loc+whichrow*bytes_per_line;
      hilite_on=TRUE;
      temp[0]=w[topwnum].xwork;
      temp[1]=w[topwnum].ywork+whichrow*gl_hchar;
      temp[2]=temp[0]+10+6*gl_wchar;
      temp[3]=temp[1]+gl_hchar;
      inv_color(temp);
      return(1);
     }

/*now calculate the address of location pointed to and coords of corresponding
  screen characters */
if (po_char)=0) addrl=w[topwnum].loc+ whichrow*bytes_per_line+whichcol;
   else  addrl=w[topwnum].loc+ whichrow*bytes_per_line+whichblock*bwl+inhex/2;
xhex=w[topwnum].xwork+hexstart +(digits+1)*gl_wchar*whichblock+inhex*gl_wchar;
xchar=w[topwnum].xwork+charstart +gl_wchar*whichcol;
yhex=w[topwnum].ywork+(whichrow+1)*gl_hchar;
ychar=yhex;
if (po_char)=0)po_char=TRUE;else po_char=FALSE;
hide_mouse();
```

```
/*now calculate the rightmost character column and hexblock column visible */
lastccolvis=min(bytes_per_line-1,
                ((w[topwnum].wwork-charstart)/gl_wch-1);
lasthcolvis=min(bytes_per_line/bwl-1,
                ((w[topwnum].wwork-hexstart)/((digits+1)*gl_wchar))-1);
cursor();
while ((mask &(key_hit=evnt_keybd())) != 13)
        {ret=key_hit/256;
         if (no_char)
          switch(ret)
          {case  RARROW:
                 if( whichcol < lastccolvis)
                   {whichcol++;cursor();xchar+=gl_wchar;cursor();addrl++;
                    if( inhex(digits-2){xhex+=gl_wchar*2;inhex+=2;}
                    else {xhex+=3*gl_wchar;inhex=0;whichblock++;}
                   }
                 break;
           case  LARROW:
                 if(whichcol)
                   {whichcol--; cursor();xchar-=gl_wchar;cursor();addrl--;
                    if(inhex){xhex-=gl_wchar*2;inhex-=2;}
                    else {xhex-=3*gl_wchar;inhex=digits-2;whichblock--;}
                   }
                 break;
           case  UARROW:
                 if(ychar-w[topwnum].ywork) gl_hchar)
                    {cursor();ychar-=gl_hchar;
                     cursor();addrl-=bytes_per_line;
                     yhex-=gl_hchar;
                    }
                 break;
           case  DARROW:
                 if(ychar(w[topwnum].ywork+w[topwnum].hwork-gl_hchar)
                    {cursor();ychar+=gl_hchar;cursor();addrl+=bytes_per_line;
                     yhex+=gl_hchar;
                    }
                 break;
           default:
                 *addrl=(key_hit & mask);ch=(key_hit & mask);
                  if((ch(040)||(ch)0176))ch='_';
                  string[0]=ch;string[1]=0;
                  cursor();
                  v_gtext(handle,xchar,ychar,string);
                  sprintf(string,"%02x",key_hit & mask);
                  v_gtext(handle,xhex,yhex,string);
                  cursor();
                   if( whichcol < lastccolvis)
                   {whichcol++;cursor();xchar+=gl_wchar;cursor();addrl++;
                    if( inhex(digits-2){xhex+=gl_wchar*2;inhex+=2;}
                    else {xhex+=3*gl_wchar;inhex=0;whichblock++;}
                   }

          } /* end switch on ret */
        if (!no_char)
          switch(ret)
          {case  RARROW:
                 if( inhex(digits-1){cursor();xhex+=gl_wchar;inhex++;
                                     cursor();
                                     if(!(inhex%2))
                                       {addrl++;whichcol++;xchar+=gl_wchar;}
                                    }
                 else {if(whichblock(lasthcolvis)
                                    {cursor();xhex+=2*gl_wchar;inhex=0;
                                     cursor();
                                     whichblock++;
                                     addrl++;
```

```
                                    addr1--;
                                    whichcol++;xchar+=gl_wchar;
                                   }
                    }
                break;
        case    LARROW:
                if(inhex){cursor();xhex-=gl_wchar;inhex--;
                          cursor();
                          if(inhex%2)
                             {addr1--;whichcol--;xchar-=gl_wchar;}
                         }
                else if (whichblock){cursor();xhex-=2*gl_wchar;inhex=digits-1;
                                     cursor();
                                     whichblock--;
                                     addr1--;
                                     whichcol--;xchar-=gl_wchar;
                                    }
                    }
                break;
        case    UARROW:
                if(yhex-w[topwnum].ywork) gl_hchar)
                  {cursor();yhex-=gl_hchar;cursor();addr1-=bytes_per_line;
                   ychar-=gl_hchar;
                  }
                break;
        case    DARROW:
                if(yhex<w[topwnum].ywork+w[topwnum].hwork-gl_hchar)
                   {cursor();yhex+=gl_hchar;cursor();addr1+=bytes_per_line;
                    ychar+=gl_hchar;
                   }
                break;
        default:
/* the next three lines change memory if the input is a valid hex char */
                if((temp[2]=val(key_hit & mask))>-1)
                {if(inhex%2){temp[0]=240;temp[1]=1;}else{temp[0]=15;temp[1]=16;}
                 *addr1=(*addr1 & temp[0])|(temp[2] * temp[1]);
                 if(whichcol<=lastccolvis) /* if the corresponding character */
                   {ch=(*addr1);           /* is visible                     */
                    if((ch<040)||(ch>0176))ch='_';
                    string[0]=ch;string[1]=0;
                    v_gtext(handle,xchar,ychar,string);
                   }
                 sprintf(string,"%02x",(*addr1)&mask);
                 cursor();
                 v_gtext(handle,xhex+gl_wchar*(inhex%2),yhex,string);
                 cursor();
                if( inhex<digits-1){cursor();xhex+=gl_wchar;inhex++;
                                    cursor();
                                    if(!(inhex%2))
                                      {addr1++;whichcol++;xchar+=gl_wchar;}
                                   }
                else if(whichblock<lasthcolvis)
                                   {cursor();xhex+=2*gl_wchar;inhex=0;
                                    cursor();
                                    whichblock++;
                                    addr1++;
                                    whichcol++;xchar+=gl_wchar;
                                   }
                    }
                 break;
                }
        } /* end switch on ret */
    }/* end while */
ursor();
how_mouse();
eturn(1);
```

```
/*************************************************************************/
/* value . Returns arithmetic value if a hex char, else -1              */
/*************************************************************************/
int val(ch)
char ch;
{int i=ch & mask;
 if ((i)='0')&&(i(='9')) return(i-'0');
 i=(i|0x20);if ( (i)='a')&&(i(='f') )return(i-'a'+10);
 else return(-1);
}
/*************************************************************************/
/* wildstr. wild[i]=1 if pstring[i]=' else wild[i] is false              */
/*************************************************************************/
wildstr(pstring)
char * pstring;
{int i;
 for(i=0;i(strlen(pstring);i++)
 if (*(pstring+i)=='?')wild[i]=TRUE;else wild[i]=FALSE;
}
/*************************************************************************/
/*wildbyte. Given pstring,(hex chars + garbage), creat@ 'target'[] by    */
/*converting the digits to arithmetic values and make 'wild'[] =1 if either */
/*digit of a hex pair is illegal                                         */
/*************************************************************************/
wildbyte(pstring)
char * pstring;
{int i,k;
 char total,j=1;
 for(i=0;i(=strlen(pstring);i++)
      if(!(i%2)) total=0;
      k=val(*pstring+i);
      if(k(0) {wild[i/2]=TRUE;if(i%2) i++;}
         else {wild[i/2]=FALSE;total+=j*k;}
      j*=16;
      if((!(i%2))&&(i)0))target[i-1]=total;
}
/*************************************************************************/
/*                    findit                                             */
/*looks for target from 'from' to 'to', returning TRUE and address in addr*/
/*if a match is found. This is used in 'find' and 'replace'              */
/*************************************************************************/
int findit(target,wildmask,from,to,addr)
char * target;
char * wildmask;
long from,to;
long * addr;
{int snap,j;long i;char * pointer;
for (i=from;i(=to-strlen(target);i++)
    { snap=TRUE;
      for(j=0;j(strlen(target);j++)
         {pointer=i+j;
          if((target[j] != *pointer)&&(wildmask[j]=FALSE))
             {snap=FALSE;break;}
         }
      if(snap)
        {*addr=i;
         return(1);
        }
    }
return(0);

/*************************************************************************/
/*                    GET LONG ADDR VALUE                                */
/*************************************************************************/
int g_long(pstring)
```

```
char *pstring;
{long tempbase=1;
 int i,k;
 if (pstring[0]=='"') /* then it's a label */
    {for(i=0;i<no_of_labs;i++)
        if(!strcmp(labels[i].label,pstring+1))break;
     if (i==no_of_labs) return(0);
     total=labels[i].value;return(1);
    }
/* else it is assumed to be a hex number WITH NO LEADING SPACE */
 total=0;
 for(i=strlen(pstring)-1;i>=0;i--)
    {k=val(*(pstring+i));
     if(k<0) return(0);
     total+=tempbase*k;tempbase*=16;
    }
 return(1);
}
/*****************************************************************************/
/*                 double block verifier                                     */
/*checks if >=3 parameters supplied and whether they are consistant.         */
/*calculates absent parameter (if any).                                      */
/*****************************************************************************/
int block2()
{long bl_size1;int i,sum=0;
 temp[0]=g_long(te_frtext);lblock1=total;
 temp[1]=g_long(te_totext);hblock1=total;
 temp[2]=g_long(te_frltext);lblock2=total;
 temp[3]=g_long(te_toltext);hblock2=total;
 for(i=0;i<4;i++)
     if(!temp[i]) sum++;
 if(sum>1) {num_alert();return(0);} /* because there is insufficient info */

 if (!temp[0])lblock1=hblock1-(hblock2-lblock2);
 if (!temp[1])hblock1=lblock1+(hblock2-lblock2);
 if (!temp[2])lblock2=hblock2-(hblock1-lblock1);
 if (!temp[3])hblock2=lblock2+(hblock1-lblock1);

 bl_size=hblock1-lblock1;bl_size1=hblock2-lblock2;
 if((hblock1<=lblock1)||(hblock2<=lblock2)||(bl_size!=bl_size1))
     {num_alert();return(0);}        /* because info is inconsistant       */
 return(1);
}
/*****************************************************************************/
/*                  Find vacant struct                                       */
/*****************************************************************************/
int find_vacant()
{int vacant;
 for(vacant=0;vacant<=maxnumwinds;vacant++)
     if(! w[vacant].on) return(vacant);
 return(-1);
}
/*****************************************************************************/
/*                  add label                                                */
/*moving these strings around is clumsy. Re-sorting indices to the labels    */
/*would be more efficient but then there would be extra work when loading    */
/*labels (unless the indices are saved as well)                              */
/*****************************************************************************/
int add_label(temp_lab)          /* trouble if no labels ?! */
struct labitem *temp_lab;
{int i,j;
 for (i=0;i<no_of_labs;i++)
     if(!(strcmp(temp_lab->label,labels[i].label))) /* then label exists */
        { labels[i].value= temp_lab->value;/*re-sort*/ return(1);}
 if (no_of_labs==MAXLABS) {lab_alert();return(0);}
 for (i=0;i<no_of_labs;i++)
     if (temp_lab->value <= labels[i].label) break;
```

```
    if (i==no_of_labs)
        {strcpy(labels[i].label,labels[i-1].label);
         labels[i].value=labels[i-1].value;
         strcpy(labels[i-1].label,temp_lab->label);
         labels[i-1].value=temp_lab->value;
        }
    else {for(j=no_of_labs;j>i;j--)
            { strcpy(labels[j].label,labels[j-1].label);
              labels[j].value=labels[j-1].value;
            }
          strcpy(labels[i].label,temp_lab->label);
          labels[i].value=temp_lab->value;
         }
    return(1);
}
/*****************************************************************************/
/* truncate string,removing all the underline characters                     */
/*****************************************************************************/
chop(pstring)
char * pstring;
{for(i=0;i<6;i++)
     if(*(pstring+i)=='_') {*(pstring+i)=0;break;}
}
/*****************************************************************************/
/* count the labels                                                          */
/*****************************************************************************/
count_labels()
{/* check thru labels until a label '=' -1 then reset no_of_labs
 for(i=0;i<LAB_LENGTH;i++)
     string[i]=255;
 string[i]=0;
 for(i=0;i<MAXLABS;i++)
     if(!strcmp(string,labels[i].label)) {no_of_labs=i;return;}
 If you reach here you've had it */
;
}
/*****************************************************************************/
/* cursor . Show/erase the cursor                                            */
/*****************************************************************************/
cursor()
{vswr_mode(handle,3); /* xor */
 vsf_interior(handle,0); /* hollow */
 if(po_char){temp[0]=xchar;temp[1]=ychar;
             temp[2]=xchar+gl_wchar;temp[3]=ychar+gl_hchar;}
 else{temp[0]=xhex;temp[1]=yhex;
      temp[2]=xhex+gl_wchar;temp[3]=yhex+gl_hchar;}
 v_bar(handle,temp);
 vswr_mode(handle,1);
}
/*****************************************************************************/
/* Calculate numerator*1000/denominator carefully*/
/*****************************************************************************/
int fraction(numerator,denominator)
int numerator,denominator;
{long top=numerator,bottom=denominator;
 int answer= (1000*top)/bottom;
 return(max(answer,1));
}
/*****************************************************************************/
/* alerts */
/*****************************************************************************/
num_alert()
{form_alert(1,"[3][label misspelt or number wrong. | Please try again][ok]");}

wind_alert()
{form_alert(1,"[3][Too many windows open.][ok]");}
```

```
form_alert(1,
        "[3][You will have to close a window ! if you want to do this.][ok]");}

lab_alert()
{form_alert(1,
 "[3][You will have to delete a label first ! before you do this.][ok]");}

sh_lab_alert()
{form_alert(1,
 "[3][You can only have 1 ! label display window][ok]");}
```